DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

A Cloud-native Vehicular Public Key Infrastructure

Towards a Highly-available and Dynamically-scalable VPKIaaS

HAMID NOROOZI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

A Cloud-native Vehicular Public Key Infrastructure

Towards a Highly-available and Dynamically-scalable VPKIaaS

HAMID NOROOZI

Master's Programme in Computer Science
Date: May 20, 2021
Supervisor: Mohammad Khodaei
Examiner: Panos Papadimitratos
School of Electrical Engineering and Computer Science
Swedish title: En cloud-native public key infrastruktur för fordon: För ett VPKI med hög tillgänglihhet och dynamisk skalbarhet

Abstract

Efforts towards standardization of Vehicular Communication Systems (VCSs) have been conclusive on the use of Vehicular Public-Key Infrastructure (VPKI) for the establishment of trust among network participants. Employing VPKI in Vehicular Communication (VC) guarantees the *integrity* and *authenticity* of Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs). It also offers a level of privacy for vehicles as VPKI provides them with a set of non-linkable short-lived certificates, called *pseudonyms*, which are used to sign outgoing messages by vehicles while they communicate with other vehicles referred to as Vehicle-to-Vehicle (V2V) or Roadside Units (RSUs) referred to as Vehicle-to-Infrastructure (V2I).

Each vehicle uses a pseudonym for its lifetime and by switching to a not-previously-used pseudonym, it continues to communicate without risking its privacy. There have been two approaches suggested by the literature on how to provide vehicles with pseudonyms. One is the so-called *pre-loading* mode, suggesting to pre-load vehicles with all pseudonyms they need, which increases the cost of revocation in case they are compromised. The other one is the *on-demand* mode, suggesting a real-time offering of pseudonyms by VPKI at vehicles request e.g., on starting each trip. Choosing the on-demand approach imposes a considerable burden of availability and resilience on VPKI services.

In this work, we are confronting the problems regarding a large-scale deployment of an on-demand VPKI that is *resilient*, *highly available*, and *dynamically scalable*. In order to achieve that, by leveraging state-of-the-art tools and design paradigms, we have enhanced a VPKI system to ensure that it is capable of meeting enterprise-grade Service Level Agreement (SLA) in terms of availability, and it can also be cost-efficient as services can dynamically scale-out in the presence of high load, or possibly scale-in when facing less demand. That has been made possible by re-architecting and refactoring an existing VPKI into a cloud-native solution deployed as microservices.

Towards having a reliable architecture based on distributed microservices, one of the key challenges to deal with is *Sybil*-based misbehavior. By exploiting Sybil-based attacks in VPKI, malicious vehicles can gain influential advantage in the system, e.g., one can affect the traffic to serve its own will. Therefore, preventing the occurrence of Sybil attacks is paramount. On the other hand, traditional approaches to stop them, often come with a performance penalty as they verify requests against a relational database which is a bottleneck of the operations. We propose a solution to address Sybil-based

attacks, utilizing Redis, an in-memory data store, without compromising the system efficiency and performance considerably.

Running our VPKI services on Google Cloud Platform (GCP) shows that a large-scale deployment of VPKI as a Service (VPKIaaS) can be done efficiently. Conducting various stress tests against the services indicates that the VPKIaaS is capable of serving real world traffic. We have tested VPKIaaS under synthetically generated normal traffic flow and flash crowd scenarios. It has been shown that VPKIaaS managed to issue 100 pseudonyms per request, submitted by 1000 vehicles where vehicles kept asking for a new set of pseudonyms every 1 to 5 seconds. Each vehicle has been served in less than 77 milliseconds. We also demonstrate that, under a flash crowd situation, with 50000 vehicles, VPKIaaS dynamically scales out, and takes $\approx$192 milliseconds to serve 100 pseudonyms per request submitted by vehicles.

Keywords: Security, Privacy, Vehicular PKI, VPKI, Identity and Credential Management, Vehicular Communications, VANETs, Availability, Scalability, Resilient, Efficiency, Microservice, Container Orchestration, Cloud, Pseudonym Transition, Pseudonym Unlinkability.

Sammanfattning

Ansträngningar för standardisering av Vehicular Communication Systems har varit avgörande för användandet av Vehicular Public-Key Infrastructure (VPKI) för att etablera förtroende mellan nätverksdeltagare. Användande av VPKI i Vehicular Communication (VC) garanterar integritet och autenticitet av meddelanden. Det erbjuder ett lager av säkerhet för fordon då VPKI ger dem en mängd av icke länkbara certifikat, kallade pseudonym, som används medan de kommunicerar med andra fordon, kallat Vehicle-to-Vehicle (V2V) eller Roadside Units (RSUs) kallat Vehicle-to-Infrastructure (V2I).

Varje fordon använder ett pseudonym under en begränsad tid och genom att byta till ett icke tidigare använt pseudonym kan det fortsätta kommunicera utan att riskera sin integritet. I litteratur har två metoder föreslagits för hur man ska ladda fordon med pseudonym de behöver. Den ena metoden det så kallade offline-läget, som proponerar att man för-laddar fordonen med alla pseudonym som det behöver vilket ökar kostnaden för revokering i fall de blir kompromettterat. Den andra metoden föreslår ett on-demand tillvägagångssätt som erbjuder pseudonym via VPKI på fordonets begäran vid början av varje färd. Valet av på begäran metoden sätter en stor börda på tillgänglighet och motståndskraft av VPKI tjänster.

I det här arbetet, möter vi problem med storskaliga driftsättningar av en påbegäran VPKI som är motståndskraftig,har hög tillgänglighet och dynamiskt skalbarhet i syfte att uppnå dessa attribut genom att nyttja toppmoderna verktyg och designparadigmer. Vi har förbättrat ett VPKI system för att säkerställa att det är kapabelt att möta SLA:er av företagsklass gällande tillgänglighet och att det även kan vara kostnadseffektivt eftersom tjänster dynamiskt kan skala ut vid högre last eller skala ner vid lägre last. Detta har möjliggjorts genom att arkitekta om en existerande VPKI till en cloud-native lösning driftsatt som mikrotjänster.

En av nyckelutmaningarna till att ha en pålitlig arkitektur baserad på distribuerade mikrotjänster är sybil-baserad missuppförande. Genom att utnyttja Sybil baserade attacker på VPKI, kan illvilliga fordon påverka trafik att tjäna dess egna syften. Därför är det av största vikt att förhindra Sybil attacker. Å andra sidan så dras traditionella metoder att stoppa dem med prestandakostnader. Vi föreslår en lösning för att adressera Sybilbaserade attacker genom att nyttja Redis, en in-memory data-store utan att märkbart kompromissa på systemets effektivitet och prestanda.

Att köra våra VPKI tjänster på Google Cloud Platform (GCP) och genomföra diverse stresstester mot dessa har visat att storskaliga driftsättningar av

VPKI as a Service (VPKIaaS) kan göras effektivt samtidigt som riktigt trafik hanteras. Vi har testat VPKIaaS under syntetisk genererat normalt trafikflöde samt flow och flash mängd scenarier. Det har visat sig att VPKIaaS klarar att utfärda 100 pseudonym per förfråga utsänt av 1000 fordon (där fordonen bad om en ny uppsättning pseudonym varje 1 till 5 sekunder), och varje fordon fått svar inom 77 millisekunder. Vi demonstrerar även att under en flashcrowd situation, där antalet fordon höjs till 50000 med en kläckningsgrad på 100. VPKIaaS dynamiskt skalar ut och tar $\approx$192 millisekunder att betjäna 100 pseudonymer per förfrågan gjord av fordon.

Nyckelord: Säkerhet, personlig integritet, identitet- och behörighetsuppgifter, tillgänglighet, skalbarhet, motståndskraftig, effektivitet, moln, pseudonymitet, anonymitet, ospårbarhet.

Acknowledgments

To begin with, I would like to express my great appreciation to my supervisor, *Mohammad Khodaei*, for his patient guidance, useful critiques and enthusiastic encouragement throughout this and earlier research works. Thank you for your supervision and helping me develop my research skills. It has been an interesting journey and very fruitful for me.

I would like to express my deep gratitude to *Prof. Panos Papadimitratos* for letting me be part of the Networked Systems Security (NSS) group and giving me the opportunity to be involved in programs and events such as CyberSecurity and Privacy (CySeP) summer school. It has been truly an honor to work with him and learning from his valuable constructive suggestions. Thank you for giving me the chance to have such an amazing experience.

I would also like to thank my dear friend, *Mateusz Mojsiejuk*, for helping me with the translation and proofreading of the Swedish abstract.

Last but not least, I am thankful to my caring partner, *Parisa*, my lovely parents, *Iraj* and *Mansoureh* and my brother, *Taha* for all of their unconditional support throughout my career.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

Within the recent years, the automotive industry has been pushing hard to leverage technological advancement to facilitate emerging of smarter vehicles [1, 2, 3, 4]. Such advancements in distributed systems have been playing a fundamental role in paving the way towards smart vehicles. The standardization bodies [5, 6, 7, 8, 9] have been trying to come up with homogenizing conventions [10] in order to better exploit the power of distributed systems and provide a foundation for the industry to build upon.

In Vehicular Communication Systems (VCSs), vehicles are supposed to communicate with each other, to which we refer as Vehicle-to-Vehicle (V2V) communication. There are also fixed entities on the road, called Roadside Units (RSUs), which are part of the underlying infrastructure. The communication between vehicles and RSUs is referred to as Vehicle-to-Infrastructure (V2I). These communications happen in the form of vehicles sending out beacon messages of types Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs) frequently. By aggregating such information, one can improve human safety to a great extent through building various applications. One example can be a collision avoidance application so that drivers would be notified if a vehicle is close to hit an obstacle. Optimizing traffic to support better mobility for vehicles under normal circumstances and/or emergency situations can also be a great application of VCSs.

In order to deploy VCS, one needs to evaluate its security and user privacy. In fact, we need to ensure the communication integrity, message non-repudiation and accountability. At the same time, the system should protect

users' privacy [11, 12, 13, 14]. In order to achieve that, there is a consensus towards deploying a Vehicular Public-Key Infrastructure (VPKI), e.g., [15, 16], which leverages the Public Key Cryptography (PKC). Vehicles are registered in the system and provided a set of *pseudonyms*, which are short-lived and anonymized certificates. Vehicles use those pseudonyms to broadcast their messages, i.e., each vehicle signs a message with a private key and receiving vehicles would verify it with the public key of that pseudonym. Since the pseudonyms are conditionally anonymized (as VPKI typically offers the ability to revoke anonymity/psuedonymity and thus, unlinkability is achieved), user's privacy is protected.

One of the main differences between the VPKI and existing Public-Key Infrastructures (PKIs) for the Internet is the number of credentials that it has to issue. In case of the Vehicular Communication (VC) system, the VPKI is required to issue pseudonyms 5 orders of magnitude more than the number of certificates that existing PKIs issue currently [15, 17]. It is worth mentioning that if we consider the Intelligent Transport System (ITS) ecosystem with pedestrians and cyclists, Location Based Services (LBSs) [7, 18, 19, 20, 21, 22, 23], and vehicular social networks [24], this number grows even further. Thus, we need to design a VPKI system so that it is capable of issuing credentials for such a system satisfying the requirements for having a large number of users.

In VCSs, it is envisioned that vehicles could interact and get pseudonyms for a large period of time. For example, it is proposed to provide vehicles with their needed pseudonyms for 25 years [25]. However, if one needs to deploy such a system and issue pseudonyms for such a long period of time, then the system becomes computationally costly and inefficient in terms of pseudonym utilization [16]. When it comes to the certificate revocation and distribution of Certificate Revocation Lists (CRLs) [26, 27, 28], it is shown that the distribution of a very large CRL among all users becomes inefficient as a large portion of the CRL is not needed for all the vehicles in the system [28, 29]. In fact, it would be waste of bandwidth for the CRL distribution, if not effectively distributed. The alternative solution here is to issue certificates for the vehicles in an on-demand manner: vehicles ask to get pseudonyms when they need them, e.g., once and twice per day. This requires having a VPKI system that can handle issuing certificates in an on-demand way. Also, as it is shown in the literature, the VPKI system could be under a Denial of Service (DoS) attack [30, 16]. In this situation, the VPKI becomes completely unresponsive, i.e., vehicles cannot renew their pseudonyms. The other scenario is a *flash crowd* [31] situation in which there is a surge in pseudonym acquisition re-

quests during rush hours. In such a situation, the VPKI system could become unreachable, or its performance drastically decreases. In short, we need to ensure that the VPKI system scales according to the rate of receiving requests and is capable of serving vehicles efficiently.

Unavailability of VPKI leads to degradation of road safety as well as user privacy. For example, an adversary could perform a Distributed DoS (DDoS) attack on the VPKI, e.g., [30, 16], in order to prevent other vehicles from getting the latest version of the CRL, or validating the revocation status of pseudonyms through Online Certificate Status Protocol (OCSP) [32]. Note that it is insecure to sign CAMs and DENMs with the private key, which corresponds to a pseudonym that is already expired. Alternatively, in case of running out of pseudonyms, a vehicle can use the private key corresponding to its Long Term Certificate (LTC); but this would harm user privacy as the LTC discloses the actual identity of the user.

As a solution when a vehicle does not have a pseudonym and cannot refill its pseudonym pool, one can use alternative anonymous credentials schemes, e.g., [33, 34, 35, 36]. In such schemes, a vehicle is registered with a Group Manager (GM) and has a group private key while it shares a group public key with all other vehicles. A message is signed by a distinct group signing key; any recipient would verify the message with group public key. In case of having no pseudonyms, each vehicle generates a pair of public and private key for itself and signs it with its group signing key, e.g., [33, 34, 35]. However, as discussed in [36], one can look at the pseudonyms issuer identities and filter out pseudonyms, signed with a different issuer identity, in this case the group signing key. Even though one can improve the user privacy by leveraging a scheme like [36] by asking vehicles to randomly changing between using their actual pseudonyms (issued by the VPKI) and the ones generated by themselves, it still degrades the performance of the safety-related applications. As described in [36], by leveraging anonymous authentication scheme for the majority of vehicles, it would result in 30% extra cryptography computation overhead in order to validate CAMs and DENMs. All in all, it is paramount to ensure that the VPKI system is highly-available, scalable, and resilient to attacks so that it could efficiently issue pseudonyms to the vehicles in an on-demand manner [37, 38].

Another important aspect of the VPKI system is that we need to mitigate Sybil [39] attacks; this happens when a vehicle has multiple valid pseudonyms at the same time. The result of such a misbehavior is that the adversary with many pseudonyms would inject multiple erroneous messages, e.g., hazard notifications, as if they were originated from multiple vehicles; alternatively, the

adversary could affect protocols which are based on voting, by disseminating authenticated false information. This is mitigated in SECMACE [16]: each vehicle is registered with one Long Term CA (LTCA) and it cannot receive more than one pseudonym from its desired pseudonym provider (Pseudonym CA (PCA)). Also, there are timing policies that prevents a vehicle from obtaining multiple pseudonyms with aligning the pseudonym lifetimes. However, when one deploys such a system, e.g., [40, 41], using microservices where services can scale horizontally to cope with the load, a malicious client could repeatedly fetch pseudonyms; this happens because each request can potentially be routed to different replicas of a microservice. As a result of this, the malicious user could receive multiple simultaneously valid pseudonyms. One can implement a centralized database, shared by all replicas and make sure about the atomicity of all transactions. But the disadvantage of such a strategy is that it drastically diminish the efficiency and performance of the VPKI system. Thus, the timely pseudonyms provisioning for large-scale mobile systems is at stake.

1.2 Research Question

To the best of our knowledge, realization of the VPKI in a large-scale deployment received limited attention. Challenges such as high availability, dynamic scalability and elasticity in a production-ready environment are not cleared. Addressing such problems often touches design patterns of the software and affecting its architecture at system level in a way that major refactoring [42] might be necessary. While refactoring a software, one should re-evaluate the processes and protocols based on which the software has been designed to make sure architectural modifications are not affecting the soundness of protocols or exposing the software to new threats.

Prevention of Sybil attacks is usually one of the challenges to confront in designing horizontally scalable systems. Minimizing the performance penalty while mitigating Sybil attacks is a question to answer in this work. Another problem to address in deployment of a large-scale VPKI is how to keep secrets safe during and after deployment and to ensure only designated microservices can access them. An investigation of secret management systems available in public cloud providers and their integration with services running on Kubernetes is done as part of this work.

Towards that direction, it is worth to explore and pave the path towards a highly available and dynamically scalable VPKI system leveraging modern tools and public cloud infrastructure and services. In order to facilitate the

deployment of a cloud-native VPKI as a Service (VPKIaaS), a great deal of automation in terms of continuous integration and continuous deployment are needed. In addition to that, the experiments that we run within the study, can bring some value in answering questions regarding capacity planning and budgeting needs when it comes to large-scale deployment of the VPKI system.

1.3 Contributions

In this thesis[1], a state-of-the-art VPKI system has been re-factored and re-architected into microservices and migrated to Google Cloud Platform (GCP) in order to achieve high-availability and dynamic scalability. Tooling have been developed around the system to smooth the way for building containerized applications and the infrastructure to run them on GCP. Also, Kubernetes resource definitions are created using a declarative language in order to facilitate deployment of microservices on Google Kubernetes Engine (GKE).

We designed a hybrid solution based on an in-memory key-value data store (Redis [44]) and a relational database (MySQL) to prevent Sybil attacks on VPKI [16] while keeping its performance intact. Following best practices, we configured microservices to scale in and out dynamically according to the load of requests on them. In order to evaluate the performance of VPKIaaS, a VPKI client was simulated and integrated with the load generation tool, Locust [45], so that we could run various stress tests leveraging synthetic load generation.

1.4 Definitions and Key Concepts

In this section, we describe the relevant terminologies of technology stacks or protocols that are used within the writing.

- **Managed Service:** A service offered by a Managed Service Provider (MSP) via ongoing monitoring, maintenance and support for customers, accompanied by a Service Level Agreement (SLA). Managed services often provide support for scalability and customers are charged using a pay-as-you-go scheme.

- **Container:** A unit of packaged software along with its dependencies running as an isolated process. Containers are lightweight bundles of an

[1]Within the context of this thesis project, we published a demo/poster [40], and a poster [43] at the ACM Wisec 2018. Also, the result of this thesis was presented at ACM Wisec 2019 [17].

operating system's user land which has become a new way of building and shipping software [46]. While running, they share the Linux kernel of their host.

- **Docker:** A software facilitating build, shipment and running containers. The features in Linux kernel has made creation and execution of containers possible, and Docker has made it easier to use those features by providing a daemon for container run-time management and tools to facilitate creation of container images and interact with them [47].

- **Kubernetes:** An open-source container orchestration platform offering services like scheduling, networking, monitoring and life-cycle management of containers [48]. It provides automation and tooling for deployment and scaling of containerized application. Kubernetes needs a container engine like Docker and it can be itself installed as a set of containers on multiple hosts forming a cluster. A Kubernetes cluster consists of a set of hosts taking part in the control plane, meaning that they responsible for management of Kubernetes components itself. It also consists of a set of machines referred to as nodes, which are responsible for running workload of the containerized applications.

- **GKE:** A managed Kubernetes cluster offered by GCP. Since installation, maintenance and life-cycle management of a Kubernetes cluster might be a tedious task, cloud providers often provide a *Kubernetes As A Service* solution, where they offload their users by taking care of the Kubernetes control plane. Running workloads on GCP, it make sense to leverage GKE as it integrates well with other services on GCP like external load-balancers [49].

- **Pod:** The smallest unit of execution in Kubernetes which may contain one or more containers. From the Kubernetes perspective, all containers in a Pod, are local to each other, so they share network and storage resources with each other. They are co-scheduled and co-located, meaning that they scale in and out together [50].

- **Deployment:** A resource object in Kubernetes defining a Pod's life-cycle and its attributes. The Kubernetes deployment controller reads the deployment, and creates or updates its state accordingly. A deployment definition can contain the desired state of Pods. Deployments can also be helpful for operations such rollout and rollbacks [51].

- **Service:** An abstract resource in Kubernetes defining a logical set of Pods, and the way they can be accessed. It can be seen as a logical load-balancer in front of a subset of Pods [52].
- **Ingress:** An Application Programming Interface (API) object at Kubernetes edge network handling external access to a service in cluster [53]. It can be seen as an external load-balancer that has the capability of terminating Secure Sockets Layer (SSL). Cloud providers use this API to create their own external load-balancer and route traffic towards Kubernetes services through that.
- **Kubelet:** A primary agent of Kubernetes, running on worker nodes which is responsible for scheduling Pods on the node and reporting their state by constantly monitoring them. It also has the responsibility of the bootstrapping the node by registering it to the Kubernetes API server. More information about Kubelet can be found at [54].
- **Horizontal scalability:** The ability of increasing/decreasing capacity by adding/removing replicas, nodes to/from a system running the same software. If a software is running on a Pod, adding more Pods with the software, is called scaling-out and shrinking the number of Pods, is the act of scaling-in.
- **Vertical scalability:** The ability of increasing/decreasing capacity by adding/removing hardware component to/from a system. If a software is running on a Pod, allocating more resources to that Pod, is the act of scaling-up and cutting resources from the Pod, is the so called scaling-down action.
- **Microservices architecture:** An architectural style for an application defining it as loosely coupled services that can scale in/out independently. According to [55], microservices design helps to grow the number of teams/people working on a big project by a systematic divide-and-conquer approach. It also lessens the risk of having bottle-necks, or making them easier to find and address. In addition to that, minimizing the blast radius in case of failures is achievable by adopting the microservices style.
- **Sybil attack:** A type of attack where an adversary exploits a system by creating more [pseudonymous] identities than it should and uses them to gain more influential advantage hence undermining the reputation of the system [39].

- **Redis:** A high performance in-memory key-value data store. Redis uses memory to store data, hence it is a very fast data store comparing to solutions where data is kept on disk. It structures data as key value pair. Comparing it to traditional relational databases, it can be seen as a single table database which consists of only two columns, where the first column is the key. GCP offers Redis in a managed-service form, called *Memory-Store*.

Chapter 2

Related Work

There has been extensive literature review on security and privacy for VC systems. There exist multiple research and industrial projects that have been investigating the security requirements of such systems. These projects focused on different aspect of the system, e.g., the SeVeCom project [56, 57], the PRESERVE project [58] and the the Crash Avoidance Metrics Partnership Vehicle Safety Consortium (CAMP VSC3) project [15]; these projects were investigating how to implement security and privacy preserving features and how to bring such a system more closely to the market via Field Operational Testing (FOT). There are standards specifying the basic requirements for the entities in the VC systems, like [5, 6, 7, 8, 9].

In Vehicular Ad-hoc Network (VANET), each vehicle is registered to a VPKI that contains multiple entities. These entities are responsible for user/vehicle enrollment, issuing them (long-term and short-term) credentials, providing CRL and OCSP. According to the standardization documents (IEEE 1609.2 WG [5] and ETSI [7]), the VPKI should be designed with separation of duty in order to preserve user privacy. Following that, separation of duty has been part of the initial design [11, 14, 59] and fully observed in the start-of-the-art identity and credential management systems, e.g., SECMACE [30, 16] and SCMS [15, 25], as well as in other architectures for security and privacy for participatory sensing [60, 61, 62].

In SECMACE [16], on which this work is relying, there is a Root Certification Authority (CA) which generates certificates for other entities in the VPKI, and not for the vehicles. In general, there are three main entities (CAs) that shaped the VPKI systems: the LTCA, the PCA, and the Resolution Authority (RA). The LTCA is responsible for user/vehicle enrollment; in fact, it generates long-term certificates for the vehicles. The PCA issues short-term

certificates, known as *pseudonyms*. Note that vehicles disseminate CAMs and DENMs using pseudonyms as their identities. The last element of the VPKI is the RA, which is responsible for doing a pseudonym resolution if needed. This action is only initiated under certain circumstances; for example, in case of an accident, the police/court could request for that.

Vehicles need to change their pseudonyms frequently. This is in fact for the purpose of user privacy protection as a vehicle trajectory cannot be tracked if it keeps changing its pseudonyms. There are several strategies for obtaining these pseudonyms [38]; one strategy is to fetch many pseudonyms and store them on the vehicle. Then the vehicle would switch from one to another frequently. Once all of them are used, it would request the VPKI for a refill. As it is mentioned in [38], such an approach is less efficient in terms of credential usages. Also, in case of revocation, the number of pseudonyms serial numbers to be included into the CRL will be considerably high. Another more efficient way is the on-demand pseudonym acquisition in which a vehicle connects the VPKI to get pseudonyms whenever needed. Ideally, a vehicle has only one pseudonym at each time. This is efficient in terms of pseudonym usage and revocation [38]; however, the downside could be the system requires reliable connection to the VPKI.

Another remark regarding the two pseudonym policies is that when a vehicle stores multiple overlapping pseudonyms in its storage, it can perform a Sybil attack [39]. In fact, that vehicle can sign multiple messages (CAMs) with different pseudonyms. Since the VPKI generates pseudonyms in an unlinkable fashion, a malicious vehicle can pretend to be multiple vehicles and the signed messages are interpreted as if they come from a number of different vehicles. This undermines the security of the system. An example of exploitation is in voting-based applications where a malicious vehicle can gain influential advantage by participating in the voting more than once. SECMACE addresses this problem through issuing the pseudonyms with non-overlapping intervals. As a result of this policy, any vehicle in the system can only have one pseudonym at any point in time. In the original form of the system [16], Sybil protection is achieved since there is only one instance of a Virtual Machine (VM) running a VPKI entity. That means the system can only scale vertically which obviously has its own limitations. However, if one needs to scale the system out by creating multiple VMs to handle more requests, it becomes vulnerable to Sybil attack. The solution to the aforementioned problem, is one of the contribution of this master thesis: by proposing an architecture using both relational and non-relational databases, we protect the system against Sybil attacks without sacrificing the performance in serving requests. More

on this can be read in Chapter 6.

Similar to the Internet, in VANET also a vehicle's private key could be compromised or the sensors of a vehicle might start broadcasting wrong information due to malfunctioning. Therefore, the system should construct a CRL [28, 63], and distribute it among all the nodes. Alternatively, it can facilitate access to check if a certificate is expired through OCSP [30]. It is crucial for VPKI to decide how to notify nodes that some of the certificates are revoked. A state-of-the-art certificate revocation system is proposed in [28, 63]: vehicles obtain the CRL which correspond to their trip duration, and not the entire CRL. Such an efficient way of distribution helps the vehicles to obtain the CRL faster, thus the vulnerability window is smaller (comparing to other approaches, for example [64, 65, 66]).

In VCSs, vehicles broadcast their CAMs and DENMs without encrypting them. As a result, one might be able to look at the messages and link them based on their attributes. Two methods for linking pseudonyms are discussed in the literature. The first one is syntactic linking [43]: an adversary can link two CAMs/DENMs by their identifiers, i.e., their pseudonyms. That means when a vehicle changes its pseudonym, an attacker can link its previous pseudonym to the new one. The second way of linking is semantic linking: an attacker looks at the data in the messages, for example, location, velocity, time, acceleration, and tries to link them. In order to mitigate syntactic and semantic linking attacks, the literature propose to use Mix-zones [67]: Vehicles enter an encrypted region, for example an intersection, and when they are not observable (i.e., in the encrypted region), they change their pseudonyms. However, as it is shown recently in [68], an adversary can link the pseudonyms with quite high probabilities when mix-zones are in place. As a mitigation strategy to make the linking even more difficult than before for an observer in a mix-zone, [69, 68] propose to distribute *chaff* CAMs and chaff DENMs for the vehicles exiting a mix-zone. Even though studies show that introducing these extra messages would add some communication delay and computation overhead, they provide a better user privacy. As a result, the syntactic and semantic linking attacks would be very difficult to exploit.

Beyond the standardization approach, i.e., vehicles leverage pseudonyms and public key crypto-systems, there are also other proposals, suggesting other primitives. For example, in the scope of VCS, vehicles could use Group Signatures (GS) [33, 34, 35, 70, 71]. With such approaches, vehicles have a common group public key and each of them has a unique group signing key. Messages are signed with the group signing key and they can be verified using the group public key. With this, the signer of a message remains anonymous; however,

the group manager can identify the actual identity for a signature. As it is shown in various research areas, GSs comes with an expensive computation cost [35, 36]. As a result, using such anonymous credentials in VANET is questionable due to such computation overhead.

Chapter 3

System Model

3.1 Overview and Assumptions

The VPKI is a system, containing multiple entities as CAs serving vehicles for different purposes. The top-level CA is Root CA (RCA) who is taking care of establishment of trust between multiple domains and will be used to verify the authenticity of lower-level CAs. A domain is defined as a region where vehicles fall under the same administrative rules and regulations [72]. Vehicles interact with LTCA in order to obtain LTC. The LTC is an X.509 certificate according to the standard [73]. Then, the vehicles that need to obtain pseudonyms use the LTC in interaction with PCA in order to obtain pseudonyms. Pseudonyms are having a relatively shorter life time. The shorter their life time is, the more difficult it will be for attackers to link them, which translates to higher privacy for the vehicles. But that also means they need to ask VPKI for pseudonyms more frequently. Also, changing pseudonyms more often, imposes a communication overhead to the system [11, 12, 38].

Figure 3.1 is an illustration of a secure and privacy preserving VC. Vehicles broadcast CAMs and DENMs all the time. These are the information that help other vehicle learn about the status of each vehicle and environment. All these messages are digitally signed and broadcasted. In the Figure, we can see three different domains and in each domain, we have the corresponding VPKI entities, i.e., LTCA, PCA, RA. In order to make the system operable, we have RCA on top of two domains establishing the trust between them. Alternatively, two domains can have certificates cross certified to establish trust. Each vehicle that broadcasts a CAM, signed with its private key, also attaches its pseudonym to it. Any vehicle nearby that receives the message has to validate it. Based on the trust establishment, we can be sure that any vehicle

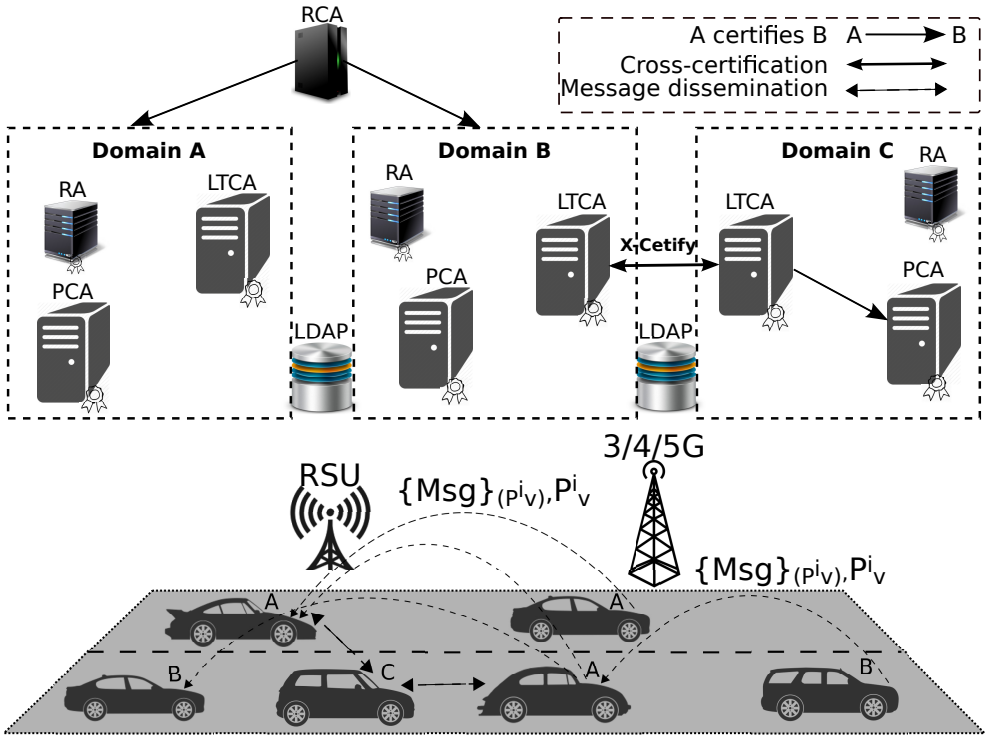


Figure 3.1: A VPKI Overview for Multi-domain VC Systems
[Taken from [17]]

can validate another vehicle's pseudonym. After the certificate validation, the message itself needs to be verified. The vehicle extracts the public key of the sender from its pseudonym (after being verified) and validates the signature. A vehicle also needs to make sure that the pseudonym is not revoked. In order to do that, it can search for it in its CRL, e.g., [28, 63]. If there is a delay in the CRL distribution, it can also query the PCA who has issued the certificate and is responsible for its revocation. Beside what is mentioned, a vehicle can use OCSP [32] to validate a pseudonym. An implementation of OCSP in the context of VC has shown [30] that it can be efficient and scalable.

For obtaining pseudonyms, there are different policies and approaches discussed in the literature [38]. Each vehicle would decide the time of obtaining pseudonyms when it runs out of pseudonym. It can send a single request and get multiple pseudonyms, or it can send multiple requests and obtaining each pseudonym in a different request. This is a trade off between linkability and practicality. For a detailed discussion, one can read [38, 16]. In this project, we assume that each vehicle can query the VPKI system and obtain pseudonyms for its desired interval. All the pseudonyms that are issued by the VPKI have

a lifetime and they are aligned by the VPKI clock. In fact, with this policy the VPKI aims to issue pseudonyms that cannot be distinguishable by only their lifetime. This would enhance user privacy. Vehicles also have Hardware Security Modules (HSMs) to protect their private keys. We also assume that a misbehavior detection system is in place for the situations that a malicious entity tries to attack the system. If an incident happens, the RA is responsible to trace it and figure out who was the malicious node in the system. This will be carried out in a way that RA needs to interact with the PCA and LTCA. As a result, if it decides to revoke some certificates, it asks PCA and the PCA constructs a CRL and distributes it. A state-of-the-art CRL distribution [28] shows that this can be done efficiently. In addition to what is stated before, we assume that the cloud providers are honest and they provide services following their SLA. The cloud providers, in our case GCP, should be trustworthy for managing the secrets of the VPKIaaS.

3.2 Adversarial Model

The adversarial model in secure vehicular communication systems has been on fully-trusted security infrastructure entities [11]. But according to the standard, we need to consider the entities *honest-but-curious* [16]. An entity is called honest-but-curious if the entity follows all rules and policies, but also tries to gather information. Later, it can interpret such information and if possible, harming user privacy. The VPKI system in [16] is designed with such a goal.

In this project, a malicious PCA could try to:

- issue many pseudonyms, potentially all valid at the same time, for a legitimate vehicle
- issue a set of pseudonyms for a non-existing vehicle
- fraudulently accuse another vehicle when it comes to resolving a pseudonym

Similarly, a malicious LTCA could try to:

- Issue a fake/invalid ticket
- fraudulently accuse another vehicle during the resolution process

The RA could also try to repeatedly ask the PCA to obtain information of pseudonyms to link them.

3.3 Requirements

The security and privacy requirements for a VC system can be found in [11]. In order to see an extensive requirements specifically for VPKI system and CRL distribution, we refer to [16, 28]. In this project, we aim to prevent Sybil attack against a VPKI system when being deployed in a scalable manner. However, we want to make sure that such a solution would not degrade the performance of the system, notably when the VPKI system issues pseudonyms. Moreover, the deployed VPKI system on the cloud should be highly-available and dynamically-scalable. The system *dynamically* scales out or scales in with respect to the load that pseudonym acquisition requests puts on it. As a result of this, the VPKI system will handle desired demanding load through systematically allocating and deallocating resources.

3.4 Security Protocols

Interactions with VPKI boils down into a couple of protocols designed specifically to facilitate the functionality the system entities are supposed to provide. In this section, we are going to cover two selected protocols that can shed some lights on how misbehaving vehicles can exploit the system. Then, in section 4.3, we propose replacement protocols to address the problems. Table 3.1 contains references to the notations used in the protocols.

Table 3.1: Notation used in the protocols
[Taken from [17]]

$(P_v^i)_{pca}$, P_v^i	a pseudonym signed by the PCA
(LK_v, Lk_v)	long-term public/private key pairs
(K_v^i, k_v^i)	pseudonymous public/private key pairs
$Id_{req}, Id_{res}, Id_{ca}$	request/response/CA unique identifiers
$(msg)_{\sigma_v}$	a signed message with the vehicle's private key
N, Rnd	nonce, a random number
t_{now}, t_s, t_e	fresh/current, starting, and ending timestamps
n-tkt, f-tkt	native ticket, foreign ticket
$H()$	hash function
$Sign(Lk, msg)$	signing a message with the private key (Lk)
$Verify(LK, msg)$	verifying a message with the public key
τ_P	pseudonym lifetime
Γ	interacting interval with the VPKI
IK	identifiable key
V	vehicle
ζ, χ	temporary variables

3.4.1 Pseudonym Acquisition Process

As it is shown in protocol 1, before pseudonym acquisition starts, each vehicle sends a request to its Home-LTCA (H-LTCA) in order to fetch an anonymous ticket to be used for its interaction with PCAs. Then it generates a Certificate Signing Request (CSR) and sends it to the PCA. The communication between the vehicle and PCA happens in a Server-authenticated Transport Layer Security (TLS) fashion so the PCA cannot yield identity of the vehicle. When the PCA receives the request, it first verifies if the ticket is signed by the H-LTCA, then it verifies the identity of the pseudonym provider, and then it generates a random number ot initiate a proof-of-possession protocol to verify the ownership of the private keys by vehicle. Calculation of a so-called Identifiable Key (IK), makes it impossible for compromised and/or malicious PCA to use a different ticket in resolution process, or to issue pseudonyms without existence of a valid ticket. The PCA can also correlate a batch of pseudonyms to a requester implicitly, enabling it to distribute CRL more efficiently. As the last step, the PCA signs the response and send it back to the vehicle.

Protocol 1 Issuing Pseudonyms (by the PCA) [Taken from [17]]

procedure IssuePsnyms(Req)
  $Req \rightarrow (Id_{req}, Rnd_{n\text{-}tkt}, tkt_{\sigma_{ltca}}, \{(K_v^1)_{\sigma_{k_v^1}}, \cdots, (K_v^n)_{\sigma_{k_v^n}}\}, N, t_{now})$
  Verify($\text{LTC}_{ltca}, (tkt)_{\sigma_{ltca}}$)
  $tkt_{\sigma_{ltca}} \rightarrow (SN, H(Id_{PCA} \| Rnd_{tkt}), IK_{tkt}, t_s, t_e, Exp_{tkt})$
  $H(Id_{this\text{-}pca} \| Rnd_{n\text{-}tkt}) \stackrel{?}{=} H(Id_{pca} \| Rnd_{n\text{-}tkt})$
  $Rnd_v \leftarrow GenRnd()$
  for i:=1 to **n do**
    Begin
      Verify($K_v^i, (K_v^i)_{\sigma_{k_v^i}}$)
      $IK_{P_v^i} \leftarrow H(IK_{tkt} || K_v^i || t_s^i || t_e^i || H^i(Rnd_v))$
      if $i = 1$ **then**
        $SN^i \leftarrow H(IK_{P_v^i} || H^i(Rnd_v))$
      else
        $SN^i \leftarrow H(SN^{i-1} || H^i(Rnd_v))$
      end if
      $\zeta \leftarrow (SN^i, K_v^i, IK_{P_v^i}, t_s^i, t_e^i)$
      $(P_v^i)_{\sigma_{pca}} \leftarrow Sign(Lk_{pca}, \zeta)$
    End
  return $(Id_{res}, \{(P_v^1)_{\sigma_{pca}}, \ldots, (P_v^n)_{\sigma_{pca}}\}, Rnd_v, N{+}1, t_{now})$
end procedure

3.4.2 Pseudonym Issuance Validation Process

In case of alerts for suspicious activities, an entity can request RA to validate the process of pseudonym issuance of the suspicious pseudonym. As shown in protocol 2, RA starts the validation process by asking the PCA who had issued the pseudonym to provide evidence for the issuance. When PCA receives the request from RA, after verification, it sends back the ticket used in the issuance procedure. Then, RA verifies the response and validates the ticket using the public key of the LTCA who issued the ticket. This ensures that not only the ticket was legit, but also the PCA could not possibly issue pseudonyms for non-existing vehicles.

Protocol 2 Pseudonym Issuance Validation Process [Taken from [17]]

$$V_j : P_v^i \leftarrow (SN^i, K_v^i, IK_{P_v^i}, t_s^i, t_e^i) \tag{3.1}$$

$$V_j : \zeta \leftarrow (P_v^i) \tag{3.2}$$

$$V_j : (\zeta)_{\sigma_v} \leftarrow Sign(P_v^j, \zeta) \tag{3.3}$$

$$V_j \rightarrow \text{RA} : (Id_{req}, (\zeta)_{\sigma_v}, t_{now}) \tag{3.4}$$

$$\text{RA} : \text{Verify}(P_v, (\zeta)_{\sigma_v}) \tag{3.5}$$

$$\text{RA} : \zeta \leftarrow (P_v^i) \tag{3.6}$$

$$\text{RA} : (\zeta)_{\sigma_{ra}} \leftarrow Sign(Lk_{ra}, \zeta) \tag{3.7}$$

$$\text{RA} \rightarrow \text{PCA} : (Id_{req}, (\zeta)_{\sigma_{ra}}, \text{LTC}_{ra}, N, t_{now}) \tag{3.8}$$

$$\text{PCA} : \text{Verify}(LTC_{ra}, (\zeta)_{\sigma_{ra}}) \tag{3.9}$$

$$\text{PCA} : (tkt, Rnd_{IK_{P_v^i}}) \leftarrow \text{Resolve}(P_v^i) \tag{3.10}$$

$$\text{PCA} : \chi \leftarrow (SN_{Pi}, tkt_{\sigma_{ltca}}, Rnd_{IK_{P_v^i}}) \tag{3.11}$$

$$\text{PCA} : (\chi)_{\sigma_{pca}} \leftarrow Sign(Lk_{pca}, \chi) \tag{3.12}$$

$$\text{PCA} \rightarrow \text{RA} : (Id_{res}, (\chi)_{\sigma_{pca}}, N{+}1, t_{now}) \tag{3.13}$$

$$\text{RA} : \text{Verify}(LTC_{pca}, \chi) \tag{3.14}$$

$$\text{RA} : (SN_{Pi}, tkt_{\sigma_{ltca}}, Rnd_{IK_{P_v^i}}) \leftarrow \chi \tag{3.15}$$

$$\text{RA} : \text{Verify}(LTC_{ltca}, tkt_{\sigma_{ltca}}) \tag{3.16}$$

$$\text{RA} : (H(Id_{PCA} \| Rnd_{tkt}), IK_{tkt}, t_s^i, t_e^i, Exp_{tkt}) \leftarrow tkt \tag{3.17}$$

$$\text{RA} : H(IK_{tkt} || K_v^i || t_s^i || t_e^i || Rnd_{IK_{P_v^i}}) \stackrel{?}{=} IK_{P_v^i} \tag{3.18}$$

Chapter 4

VPKI Services Overview

4.1 VPKI as a Service

As part of this work, we take an implementation of the VPKI system [16] and modernize it using state-of-the-art tools and technical stacks available at the moment. We leverage public cloud providers, e.g., GCP to facilitate the setup. However, it is worth mentioning the architecture is cloud-agnostic; meaning that any public, private or hybrid cloud provider can be a potential platform for offering VPKIaaS. To be able to achieve that, we migrate the workloads of VPKI into microservices running on Docker containers orchestrated by Kubernetes [48]. We use the managed Kubernetes service, GKE, offered by GCP. Figure 4.1 shows a high-level abstraction of how the VPKI entities, or the so called microservices, are being deployed.

We turn each entity of the VPKI into a deployment which defines the traits of pods running the corresponding workload. In order to do so, we create a Docker image from each of the VPKI entities, and store them in Google Container Registry, a service on top of an storage service dedicated to hold Docker or other container types images. A pod is the smallest unit of a workload that contains one or more containers. The pods report their state along with metrics to the Kubernetes controller, so that the controller triggers relevant actions according to the status changes. We define three main deployments for the entities, RA, LTCA and PCA. Each creates a set of pods where each pod is running a single container. Each pod, publishes its health and load metrics for the Kubernetes controller. If the controller does not see the health metric of a pod, or receives an un-healthy status for a pod, it acts by killing the pod and re-spawning a new one. We also deploy a resource, called Horizontal Pod Autoscaler (HPA) which sets some requirements and actions to be done when

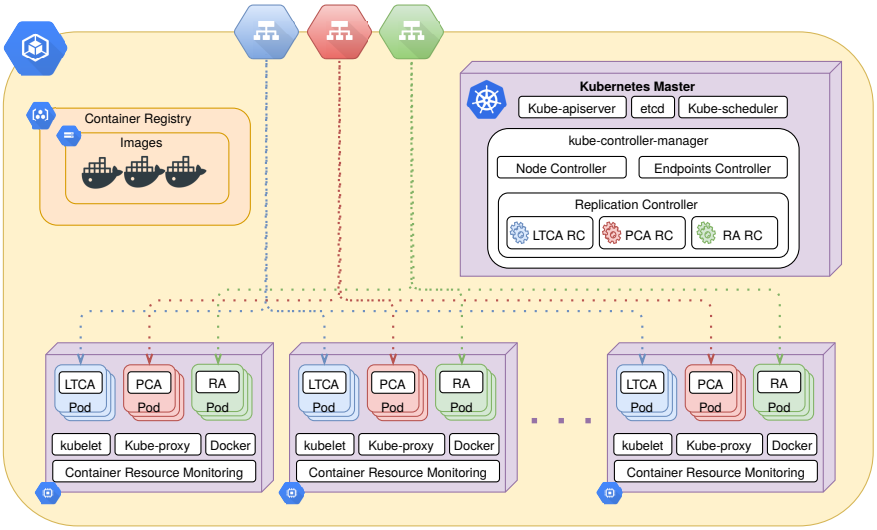


Figure 4.1: A High-level Overview of VPKIaaS Architecture [Taken from [17]]

the requirements are met. We define a threshold 60 per-cent CPU usage as the requirement to act upon, and the action of scaling out/in depending on if the CPU usage is above or below the threshold. There is an absolute minimum and maximum number of pods defined in the HPA in order to avoid undesired behavior. In order to handle the ingress requests towards the pods, we define three application load-balancers in GCP that are connected to ingress-controllers of the Kubernetes, serving requests to RA, LTCA and PCA.

4.2 Implementation

This section focuses on the implementation details of the work on VPKIaaS paving the way towards a large-scale deployment.

4.2.1 Microservices Architecture

One of the early things done with the code base was trying to identify system entities and re-architect the software into microservices [55] architecture design. Each type of CA within the system, e.g., PCA, LTCA or RA, is considered to be a microservice. Code refactoring [42] has been done to make each microservice scalable horizontally. In order to do that, software patterns like singletons [74] have been removed, and architecture of the software has been modified to make it stateless as much as possible. Such modifications

are playing a fundamental role in scalability of the system.

Using container technologies, we have built *base* container images, on top of which microservices can be built. Containers have been used both in compilation of the libraries and also installation and execution of microservices. Each microservice will be compiled into a single binary along with some configuration files that will be placed into a container being executed using Apache web server.

We implemented a functionality in each microservice that at any point in time, they can be queried about their health and load. The health metric will be used later by the scheduling service, kubelet [54], to make sure the microservice is healthy. The load metric will be used by the autoscaling service, HPA [75], to trigger scaling in/out actions according to the logic we feed into the system regarding scalability.

4.2.2 Load Generator

In order to be able to run experiments against our system, we implemented an entity representing a vehicle. We integrated our vehicle implementation with the Locust framework [45] with minor modifications so it can leverage the power of Locust to put arbitrary load on the system. The load generator is also considered to be a microservice that is deployed similar to other microservices of the VPKI system. When we deploy our load generator, using the web interface of Locust, we can set the parameters of load generation such as number of vehicles and their hatch rate. This way we can easily conduct experiments in simulating various scenarios like flash crowd [31] situation.

4.2.3 Deployment

Listing 4.1: LTCA Deployment resource

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: ltca
  namespace: ltca
  labels:
    app: ltca
    role: ltca
    tier: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ltca
      role: ltca
```

```
      tier: frontend
  template:
    metadata:
      labels:
        app: ltca
        role: ltca
        tier: frontend
    spec:
      containers:
      - name: ltca
        image: eu.gcr.io/vpki-nss-kth-220115/vpki:ltca
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "128Mi"
            cpu: "500m"
          limits:
            memory: "128Mi"
            cpu: "500m"
        livenessProbe:
          exec:
            command:
            - /opt/vpki/scripts/healthcheck.sh
            - ltca
          initialDelaySeconds: 60
          periodSeconds: 10
```

Kubernetes [48] is our chosen container orchestration platform used to run and manage the workload. To facilitate the deployment of our microservices on Kubernetes, we used the declarative method of defining resources in YAML [76] so we could version control them along with the rest of the software code. Each of the VPKI entities have four resources defining them. One is the *Deployment* resource which contains the characteristics of the Pods that are supposed to run the workload of the corresponding microservice. Listing 4.1 depicts the deployment resource of the LTCA including information about 1) memory and processing capacity, 2) the isolated namespace where the pods are supposed to be running in, 3) the Docker image of the microservice, and 4) commands that are supposed to be used for health-check monitoring.

Listing 4.2: LTCA Service resource

```
apiVersion: v1
kind: Service
metadata:
  name: ltca
  namespace: ltca
  labels:
    app: ltca
    role: ltca
    tier: frontend
spec:
  selector:
    app: ltca
    role: ltca
```

```
    tier: frontend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: NodePort
```

The second resource is called *Service*. It defines an abstract entity in front of the Pods, routing received requests to healthy Pods. Listing 4.2 is the simple Service definition we have used for LTCA.

Listing 4.3: LTCA Ingress resource

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ltca
  namespace: ltca
  labels:
    app: ltca
    role: ltca
    tier: frontend
spec:
  tls:
  - secretName: ltca-secret
  rules:
  # - host: ltca.vpki-nss-kth.se
  - http:
      paths:
      - path: /*
        backend:
          serviceName: ltca
          servicePort: 80
```

The Service resource by itself is not enough to handle incoming traffic from outside of Kubernetes, as it is just an abstraction acting as an internal Load-balancer. The *Ingress* resource is used to create an external Load-balancer and hook it into the Service. Listing 4.3 is the Ingress we defined for LTCA. Since we are using the managed GKE on GCP, the Ingress resource creates an external Load-balancer in GCP.

Listing 4.4: LTCA HPA resource

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: ltca
  namespace: ltca
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: ltca
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 60
```

The last resource that the microservice needs, is the HPA which defines thresholds for scaling in/out actions according the load metric we define. Listing 4.4 is the simple HPA defined for LTCA which is using CPU utilization as the load indicator. Listing 4.4 tells the Autoscaling service to scale out the LTCA Pods when their average CPU utilization is above 60 percent, or scale in the Pods when their average CPU utilization is observed to be below 60 percent. It also defines the maximum and minimum number of Pods in order to avoid resource exhaustion of the Kubernetes cluster.

4.2.4 Secret Management

So far, it is well understood that VPKI entities are supposed to have their own key-pairs to be able to perform their cryptographic operations. These key-pairs contain secrets that need to be protected both during the deployment of the VPKI service, and also during runtime where entities are required to load their secrets. Obviously, the secrets cannot be baked into the container images of the VPKI entities, because that turns the image into something to protect. In order to control the access to secrets needed for the VPKI entities, there are two main approaches. One is to use the Key Management Service (KMS) offered by the cloud provider, in our case GCP. The other is to use the Kubernetes secret management system. Each has its own pros and cons. The KMS from GCP offers various cryptographic algorithms and primitives, e.g., AES-256, RSA and Elliptic Curves. In addition to that, there are HSM services [77] on GCP that if used along with KMS, it guarantees a protection level in compliance with Federal Information Processing Standard (FIPS) 140-2 level 3. [78]. The cloud Identity & Access Management (IAM) service is then used to facilitate a role based access control to the keys that can encrypt and decrypt a secret.

While using the KMS and IAM services to handle the access management for secrets of the VPKI entities sounds promising, but it is a dependency to the specific cloud provider. Also calling Google APIs for every cryptographic action would be expensive in terms of latency, which does not fit into the solution we want to offer. The Kubernetes secret management is a cloud-agnostic solution. It creates volumes containing the secrets, and mounts them beside the microservices who are supposed to have access the secrets. That happens with the help of isolation that namespaces provide. besides being cloud-agnostic, another benefit of such a solution is that, accessing the secrets happens way faster than the KMS solution, as there is no need to call any API to fetch the secrets, instead the Pod needs to read a file from disk to load its secret. However, the downside would be that it does not offer any protection while deploying the services. Also no security compliance is guaranteed.

We suggest to have a hybrid approach combining the two solutions stated above. We introduce a bootstrapping phase where we generate master keys in the cloud provider's KMS for each entity. Then using the KMS keys, we encrypt the actual keys that VPKI entities are supposed to use. We create Kubernetes secrets by adding the encrypted keys beside each microservice as part of the deployment. Hence, secrets are protected during deployment. When the entities want to use their keys, they use the KMS decryption functionality and decrypt the secret volume that they have access to. The access to the KMS keys are also protected using the role based access control provided by the IAM service. Hence, the Pods have their keypairs for use and there is no more need to call the KMS API to fetch secrets.

4.3 Sybil Attacks against VPKIaaS

Distributed systems often need to address the Sybil attack problem [39]. Running multiple instances of a workload, increases both performance in terms of large scale deployment and serving more end-users, but also increases the risk of Sybil attacks against the system. In a more traditional deployment of the VPKI, where entities were not horizontally scalable, and the only scalability option was vertical, Sybil attacks were not a concern. However, when services are horizontally scalable, more than one replica of the same service would be running at a time. Obviously, there should be measurements implemented both to facilitate the coordination between replicas and to avoid Sybil attacks.

In the existing implementation of the VPKI, there exists a relational database system, MySQL, for each entity. LTCA, PCA, and RA store data of their executed operations in the databases. Every time they receive a request, they send a query to the database to check if the request has not been served before. As a result of the modernization efforts towards realization of a large-scale deployment of VPKI, services are horizontally scalable. Therefore, vehicles' requests will now get distributed among multiple replicas of the same entities to improve efficiency and performance of the system. However, this can create a risk of Sybil based misbehavior, in a way that two identical requests from the same vehicle can end up being served by two different replicas at the same time, without having a chance of deterring the double spending.

Traditionally, distributed systems first answer to Sybil type problems was to utilize the locking mechanism in relational database systems and having synchronous communication between the replicas and the database system. That of course solves the issue, but it comes with huge performance penalty.

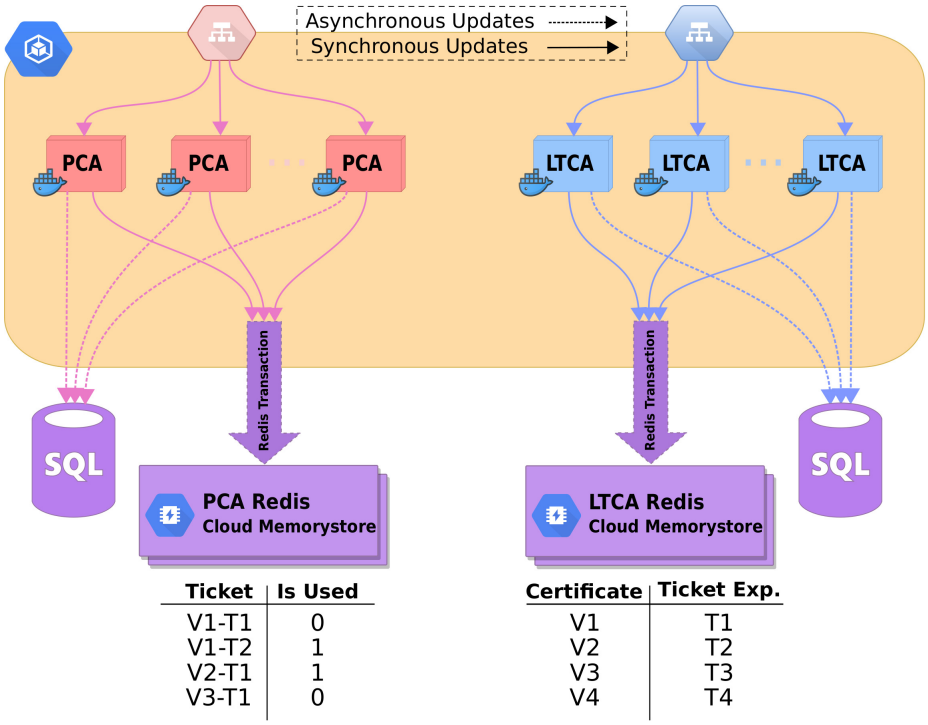


Figure 4.2: VPKIaaS Sybil attack prevention using Redis and MySQL [Taken from [17]]

One of the main reasons contributing in the performance degradation is that, having synchronous connections to the database, means that all replicas need to wait for one replica to be done with its operation so they can query values from the table. So, the database operation could be the bottleneck in efficiency of the system. Most of the relational database systems use disks as their back-end storage, and disk Input/Output (I/O) operations are expensive, specially in situations where system entities need to store multiple data records for the vehicles being served.

In order to mitigate Sybil attacks and also not sacrificing the performance, we have used a hybrid setup of relational and NoSQL databases. for each system entity of LTCA and PCA, there exist an in-memory key-value database, Cloud Memorystore, that replicas of the system entities have synchronous communication with. Cloud Memorystore is a managed Redis database service hosted by GCP. I/O operations are much faster in Redis, as data structure is stored in memory rather than disk. In addition to that, since we do not store all the data related to a vehicle in it, the waiting time for replicas are much lower. In order to store the rest of data for served vehicles, we use a relational database in an asynchronous fashion which does not degrade performance.

This way, replicas of LTCA and PCA can keep serving vehicles and remove the concern of Sybil attacks with the quick and single-threaded I/O operation that they do against their corresponding Redis instance. Figure 4.2 shows how entities communicate with their relational database and Cloud Memorystore. More details on the implementation is available at section 4.4.

4.4 Sybil Attack Prevention

Since Redis [44] is single-threaded, every command is guaranteed to be atomic. Redis also offers *Transactions* [79] in order to make sure about a sequential execution of the commands atomically. Leveraging the two, a Sybil attack prevention protocol has been implemented for LTCA and PCA.

4.4.1 Ticket Acquisition from LTCA

The Redis Key-Value table for LTCA holds the serial number of vehicles as the keys, and the expiration time of the previously issued tickets as values. Obviously, the value could be $NULL$ for a vehicle that has not got a ticket before. As shown in protocol 3, when a request is received by an instance of LTCA, the value for its serial number is fetched from Redis. If the value is $NULL$ or if the value is less than or equal to the start-time of the requested ticket, the request would be considered legitimate, otherwise, it would be rejected as a Sybil attack request. Considering the request being legitimate, the value would be updated by the expiration time of the newly received request and ticket issuance procedure would be invoked. In case of any failure in the ticket issuance procedure, the value would be reverted to the old one.

Protocol 3 Ticket Request Validation (by the LTCA using Redis), taken from [17]

procedure VALIDATETICKETREQ($SN^i_{LTC}, tkt^i_{start}, tkt^i_{exp}$)
  $(value^i) \leftarrow$ RedisQuery(SN^i_{LTC})
  if $value^i == NULL$ **OR** $value^i <= tkt^i_{start}$ **then**
    RedisUpdate(SN^i_{LTC}, tkt^i_{exp})
    $Status \leftarrow IssueTicket(\dots)$ ▷ Invoking ticket issuance procedure
    if $Status == False$ **then**
      RedisUpdate($SN^i_{LTC}, value^i$) ▷ Reverting SN^i_{LTC} to $value^i$
      return $(False)$ ▷ Ticket issuance failure
    else
      return $(True)$ ▷ Ticket issuance success
    end if
  else
    return $(False)$ ▷ Suspicious to Sybil attacks
  end if
end procedure

4.4.2 Pseudonym Acquisition from PCA

Similar to LTCA, a Key-Value table in Redis is used for PCA, where serial number of ticket is the key, and a Boolean flag is used as value. As shown in protocol 4, when a request for pseudonym is received by an instance of PCA, the value for its ticket serial number would be read from Redis. If the value is $NULL$ or $False$, it means no pseudonyms has been issued for the ticket before and the request is legitimate. Otherwise, it is suspicious to be Sybil attack and would be denied. Considering the request was legitimate, the value would be set to $True$ and the pseudonym issuance procedure would be invoked. In case of any failure in pseudonym issuance procedure, the value in Redis would be reverted to $False$.

Protocol 4 Pseudonym Request Validation (by the PCA using Redis), taken from [17]

procedure VALIDATEPSEUDONYMREQ(SN^i_{tkt})
  ($value^i$) ← RedisQuery(SN^i_{tkt})
  if $value^i == NULL$ **OR** $value^i == False$ **then**
    RedisUpdate($SN^i_{tkt}, True$)
    $Status \leftarrow IssuePsnyms(\dots)$ ▷ Invoking pseudonym issuance
    if $Status == False$ **then**
      RedisUpdate($SN^i_{tkt}, False$) ▷ Reverting SN^i_{tkt} to False
      return $(False)$ ▷ Pseudonym issuance failure
    else
      return $(True)$ ▷ Pseudonym issuance success
    end if
  else
    return $(False)$ ▷ Suspicious to Sybil attacks
  end if
end procedure

Chapter 5

Qualitative Analysis

Existing literature [28][16] have already analyzed security and privacy of VPKI entities. In this section, we cover the analysis of VPKIaaS focusing on aspects of a cloud-native deployment of VPKIaaS on a public cloud provider.

5.1 Security and Privacy Analysis

5.1.1 Trust in Cloud Provider

In general, running services in public cloud, can raise certain security and privacy concerns. Meanwhile, ideas such as Fully Homomorphic Encryption (FHE) seem to offer a brighter future regarding privacy concerns [80]. However, there is no technical approach that promises a solution against malicious cloud providers compromising availability of deployed services [81]. Therefore, it is fundamental to realize that, trusting the cloud provider is already assumed to some extent before deployment of services on it. The requirements for establishment of trust can be met by verification of compliance certifications. So legal frameworks, and complying to regulations, is considered to be the right approach in trusting cloud providers [82]. We have used GCP to deploy our VPKIaaS. However, leveraging microservices architecture and relying on Kubernetes as the platform, means that by introducing minor modifications, any public cloud provider could be used to host VPKIaaS. In a more extreme concern about public cloud providers, authorities may even choose to deploy VPKIaaS on private cloud hosted on-premises. That is feasible because almost all services offered by public cloud providers, can be hosted privately using alternatives such as OpenStack [83].

5.1.2 Sybil Attacks

A malicious vehicle can try mounting an attack by sending multiple requests asking for overlapping tickets from LTCA or trying to re-use a ticket to acquire more pseudonyms from PCA. Deploying VPKI entities as microservices that are capable of scaling horizontally can expose the system to such attacks. VPKIaaS leverages cloud Memorystore [84], a managed Redis service offered by GCP to quickly identify Sybil-based misbehavior. Since Redis is operating in a single threaded fashion, it achieves atomicity by sequentially executing all received operations. Redis *transaction* is a series of commands that can be grouped together being sent to Redis having a guarantee that either all or none will be executed atomically. Sybil protection is a configuration directive that can be toggled on and off during the deployment of VPKIaaS. Having it on, guarantees that Sybil based attacks are denied.

5.1.3 DDoS Attacks

External attackers can launch DDoS attacks against VPKIaaS by sending fake certificates to LTCA or requesting pseudonyms from PCA using fake tickets. This can lead to a signature flooding attack [85]. There are various security mechanisms in VPKIaaS to protect and/or mitigate such attacks. First is the dynamic scalability of the system, which scales out horizontally in resources automatically. Having the VPKIaaS deployed on a cloud provider, such as GCP gives us the power to potentially scale out and utilize as much resources as the cloud provider has allocated to the project which is considerably high by itself. In addition to that, since services are actually running on Kubernetes, which is the de facto platform for running microservices, VPKIaaS can be potentially expanded into a multi-cloud deployment. Obviously, aforementioned solutions are not enough to stop a DDoS attack assuming it is done by a strong attacker. They only guarantee to scale out resources, meaning that the attacker wins eventually if it has advantage in resources. However, applying built-in DDoS protection features on the public facing load-balancers can solve the problem to a large extent. Cloud Armor is the battle-tested DDoS protection system on GCP load-balancers, and it is claimed to be the one protecting Google services such as *GMail*, *YouTube* and *Google Search* [86]. That being combined with GCP Web Application Firewall (WAF) which is also extensible by adding more rules, can provide a solid safety against DDoS attacks. Introducing a dynamic puzzle-based protection, could be another security measurement on top of what is already mentioned, which is explained as part of the future work in section 7.2.

5.1.4 System Entities Failure

Software or hardware failure are also considered among the risks against availability of VPKIaaS. For each entity, we have implemented a health-check by verifying its operation internally. Each LTCA Pod, while running, executes the ticket acquisition procedure on itself periodically, and the result is only reported as a success in health-check. Similarly, each PCA Pod does a pseudonym acquisition on itself periodically. These requests are using internal functions of each service, so such requests cannot be made through the external interface of the microservice. The health-check metric is then exposed to Kubelet, which is responsible for monitoring the *liveness* and *readiness* of Pods, and in case of failing to verify them, it kills the Pod and replaces it by running a new one. Similarly, there is a health-check for Kubernetes worker nodes, which makes sure they are running the workloads, and if not, they will be drained and cordoned by the Kubernetes scheduler. Leveraging the aforementioned techniques, we can make sure to achieve high availability and fault tolerance.

Chapter 6

Quantitative Analysis

In this chapter, we will cover the steps taken to prepare an experimental environment, and go through the results of conducted experiments.

6.1 Prerequisite Setup

We started by refactoring the code base of the VPKI and made it ready to be deployed as a cloud-native software. After being done with refactoring, leveraging scripting languages, we created tools to facilitate the compilation, integration and deployment of the microservices. We also built monitoring tools to be deployed beside the VPKIaaS, and scripts to be executed on the deployer machine so we can collect arbitrary metrics while the system is live and accepting workload.

Compilation

We started by building a Docker image based on Ubuntu and adding necessary packages needed to test and compile VPKI code, such as FastCGI [87], XML-RPC [88] and Protocol Buffer [89]. We use this image as our base image for the build.

We then created scripts iterating over entities of the VPKI system that their code has been modified since the last time, loading them into a new container instantiating from our base image, compiling the code and moving the outcome to a directory hierarchy on the builder host called *packaged-build*.

Installation

Similar to the compilation step, we started by creating a slimmer base image, which has necessary packages to run VPKI entities. We also created configuration templates for each entity and place-holder for their certificates. In addition to that, steps like generating certificates, or skipping them if they are already provided through other methods are automated as well. After running the installation scripts, we will end up having a Docker image ready to run for each microservice. Then we push all the images to Google Container Registry (GCR).

Load Generator

The VPKI system, already had an entity to represent as a vehicle; however, it has been implemented as part of the system to just verify the functionality, and it could not be used in our load/stress testing experiments . So we had to rewrite another implementation of the vehicle entity in Python so that we can hook it into the Locust [45] load generator. Also, since Locust could not work with XML-RPC out of the box, we had to implement a hook for Locust so that it can load our Vehicle implementation. For the sake of experiments, we were only interested in actions of ticket acquisition from LTCA and pseudonym acquisition from PCA, so the hooks only support the aforementioned actions. Similar to other VPKI entities, we scripted the process of building the *Load Generator* container image and pushing it to GCR.

Kubernetes

As part of the effort for having automation around deployment of the VPKI system, we have scripted the creation of GKE running 5 VMs (n1-highcpu-32) each having 32 vCPUs and 28.8GB of memory. Along with that, other GCP services that are required for our system to be functional such as Cloud SQL [90] and MemoryStore [84] are created during the bootstrapping phase VPKI. We then defined VPKI microservices and resources in YAML [76] which is the declarative method of defining resources in Kubernetes. The isolated namespaces, Deployments, Services, Ingresses are definitions we need for deployment of the VPKIaaS. In addition to those, we also added similar definitions for the load generator installation.

Monitoring

In order to collect various metrics from the VPKI system while having it under load based on different scenarios, we implemented and deployed monitoring mechanisms. One of the most popular tools for monitoring Kubernetes clusters and the workload on them, is Prometheus [91], a Time Series DataBase (TSDB) that can easily run on the cluster itself, fetching metrics using the Kubernetes API. In order to visualize the metrics on Prometheus, and have visibility during the experiments, we use Grafana [92]. In order to be able to query and export data from Prometheus, we used Styx [93] that is capable of running complicated queries against Prometheus and fetching results in *CSV* format. Also, since we needed live data from HPAs as the load was changing during the experiments, we wrote our own script to monitor HPAs recording their state every second.

In addition to that, we also needed metrics from the Load Generator, Locust. At the time of doing our experiments, Locust was offering a web interface showing the outcome of requests every second. However, it did not have a way to access to historical data. So, we wrote our own tool that hooks to the Locust web interface and records the result every second.

6.2 Experiment Scenarios

In order to evaluate the performance of our VPKIaaS, we measured its efficiency in pseudonym and ticket acquisition processes under different load testing scenarios and configurations. Leveraging the Locust load generator, we managed to generate arbitrary volume of synthetic requests and send them towards the system with various configurations. Also, in order to test how the system performs in flash crowd scenario, we added a sudden and rather huge increase in number of pseudonym requests; our results show that the VPKIaaS can scale dynamically and continue to serve requests without disruption.

Table 6.1 is a reference of the configurations we used in the experiments. Config-1 is representing a configuration where 1000 vehicles join the system by the rate of one vehicle per second. After joining, vehicles keep sending pseudonym requests towards the system and wait for one to five seconds between each two requests. Multiple experiments have been conducted using Config-1 where they differ in the number of pseudonyms in each request selected from 100, 200, 300, 400 or 500 pseudonyms. Config-2 shows a flash crowd [31] situation in which the load generator has two states. First it starts by defining 100 vehicles joining the system by the pace of one vehicle per second.

Table 6.1: Experiment Parameters
[Taken from [17]]

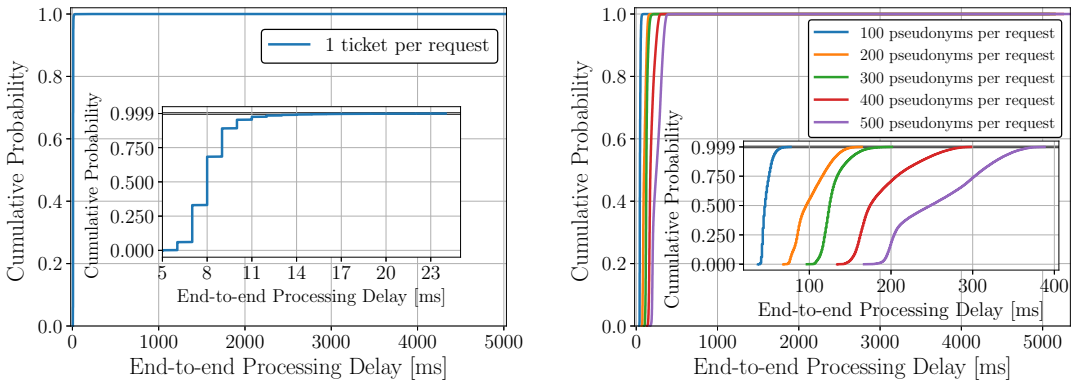
Parameters	Config-1	Config-2
total number of vehicles	1000	100, 50,000
hatch rate	1	1, 100
interval between requests	1000-5000 ms	1000-5000 ms
pseudonyms per request	100, 200, 300, 400, 500	100, 200, 500
LTCA memory request	128 MiB	128 MiB
LTCA memory limit	256 MiB	256 MiB
LTCA CPU request	500 m	500 m
LTCA CPU limit	1000 m	1000 m
LTCA HPA	1-40; CPU 60%	1-40; CPU 60%
PCA memory request	128 MiB	128 MiB
PCA memory limit	256 MiB	256 MiB
PCA CPU request	700 m	700 m
PCA CPU limit	1000 m	1000 m
PCA HPA	1-120; CPU 60%	1-120; CPU 60%

When all 100 vehicles are in the system actively requesting pseudonyms, and waiting one to five seconds between their requests, the load generator suddenly goes to the second state by increasing the total number of vehicles to 50000 and setting the hatch rate to 100, meaning that hundred vehicles join the system every second. Collecting enough data as an excerpt to show resilience, high availability and dynamic scalability of VPKIaaS, we change the load generator settings back to state one. The flash crowd scenario experiment has been repeated three times, each time changing the number of pseudonyms vehicles ask in each request chosen from the 100, 200 or 500 pseudonyms.

Considering the assumption of pseudonyms being issued with non-overlapping intervals, having 100 or 500 pseudonyms and using them for a whole day, makes each pseudonym lifetime ≈ 14 or ≈ 3 minutes respectively. Real world data-sets from Tapas-Cologne [94] and LuST [95] are suggesting that average trip duration for each vehicle is ten to thirty minutes on daily basis. Also, according to a report from United States Department of Transportation (DoT) [96], one hour is the average trip duration of a vehicle per day in the US.

6.3 Large-scale Pseudonym Acquisition

Figure 6.1.a shows a Cumulative Distribution Function (CDF) presentation of the ticket issuance end-to-end latency in milliseconds where the experiment was running using Config-1 in table 6.1. As it is shown, the LTCA service has served 99.9% of ticket requests by vehicles in 24 milliseconds. 6.1.b shows a CDF for pseudonym issuance end-to-end delay. For that, we repeated the experiment 5 times, each time different number of pseudonyms were requested. Asking for 100 pseudonyms per request, 99.9% of vehicles are served in less than 77 milliseconds. Increasing the number of pseudonyms per request, obviously introduces more latency in the system; e.g., asking for 500 pseudonyms in each request, 99.9% of vehicles could be served within 388 milliseconds. Our results confirm that VPKIaaS is capable of handling gradual increase in load efficiently and vehicles are being served within a reasonable time.



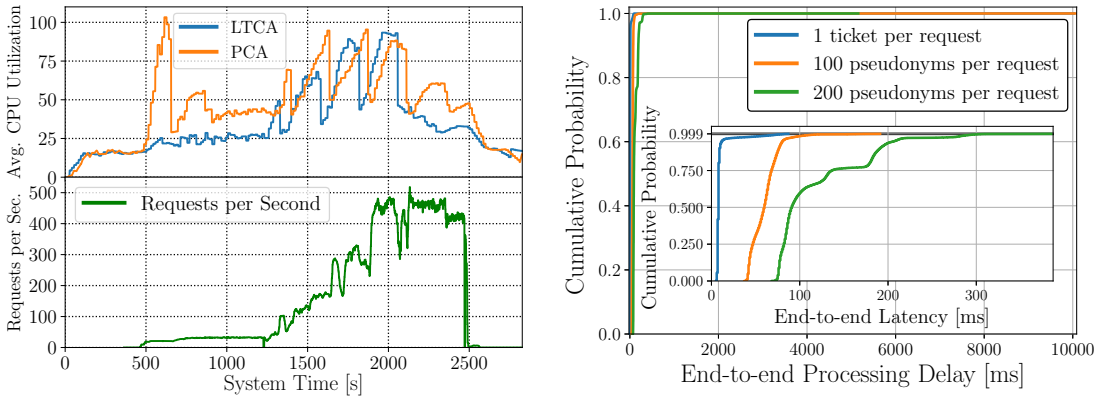
(a) Ticket Issuance

(b) Pseudonyms Issuance

Figure 6.1: (a) CDF of end-to-end latency to issue a ticket. (b) CDF of end-to-end processing delay to issue pseudonyms.
[Taken from [17]]

6.4 Flash Crowd Situation

Figure 6.2 shows how the system perform under flash crowd situation. Config-2 from table 6.1 has been used for this experiment. We have measured average CPU utilization by LTCA and PCA Pods (6.2.a top), and number of requests per second recorded by Locust (6.2.a bottom). As described in 6.2, for the flash crowd experiment, we change the load generator configuration twice to take the system to flash crowd mode and bring it back to normal. As shown in



(a) (b)

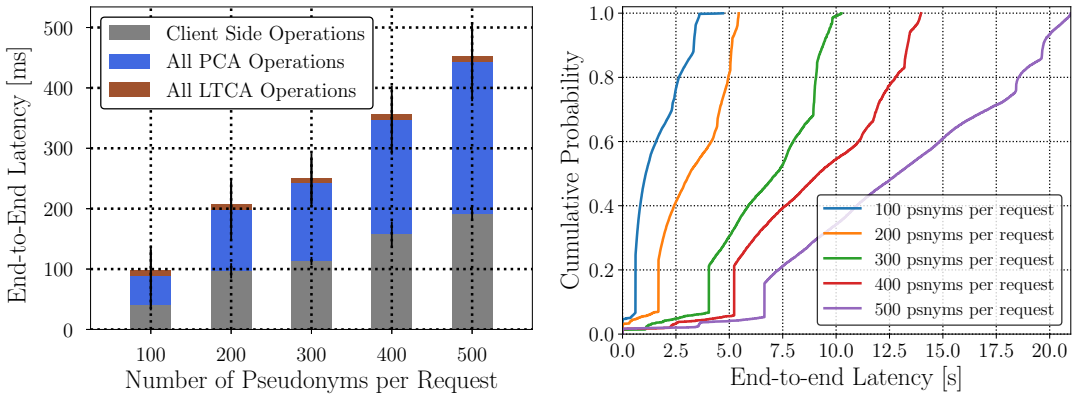
Figure 6.2: VPKIaaS system in a flash crowd load situation. (a) CPU utilization and the number of requests per second. (b) CDF of processing latency to issue tickets and pseudonyms.

[Taken from [17]]

table 6.1, we have configured HPAs [75] for LTCA and PCA setting the CPU threshold to 60% meaning that when the average CPU utilization on LTCA or PCA passes 60%, HPA creates an event asking the Kubernetes scheduler to scale out by adding more replicas to LTCA or PCA. There is a similar mechanism for scaling in replicas by removing them when their average CPU utilization reaches below 60% and HPA estimates that by removing replicas, CPU utilization would not go over 60% again. One can also define minimum and maximum number of replicas for HPA to avoid unavailability or exhaustion of services. For the sake of our experiment, we have set the minimum number of Replicas to 1, and the maximum number of replicas for LTCA and PCA to 40 and 120 respectively.

Figure 6.2.b shows a CDF for end-to-end processing delay for ticket and pseudonym issuance in flash crowd situation. As illustrated, 99.9% of tickets have been issued within 87 milliseconds. The processing delay to issue 100 pseudonyms per request for 99.9% of requests is 192 milliseconds. Obviously, comparing the result from 6.2.b and 6.1, one can conclude that, although the latency both for ticket and pseudonym issuance, has been increased in the flash crowd scenario, the delays are within reasonable range, showing that the VPKIaaS can handle sudden stress in load by quickly scaling out services and continue to serve vehicles without disruption.

Figure 6.3.a shows the delay introduced by different system entities for acquiring various batches of pseudonyms from 100 to 500 according Config-2 in table 6.1. As shown, the processing delay for issuing 100 pseudonyms is



(a) (b)

Figure 6.3: VPKIaaS system with flash crowd load pattern. (a) Average end-to-end latency to obtain pseudonyms. (b) CDF of end-to-end latency, observed by clients.

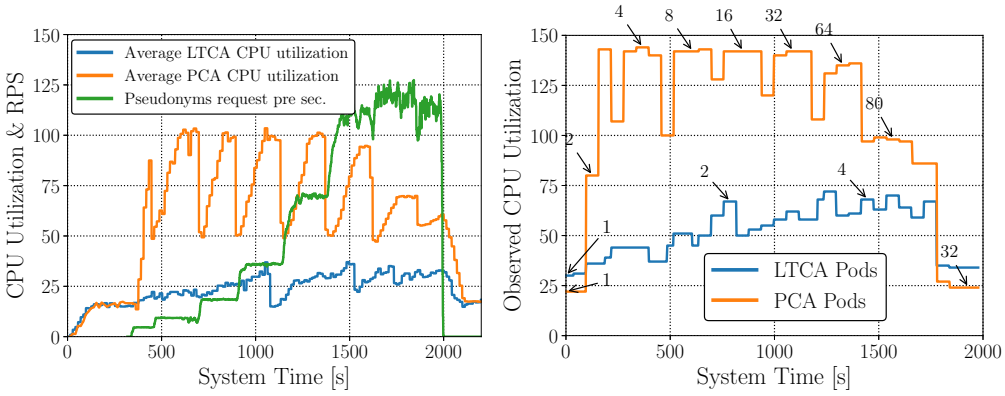
[Taken from [17]]

approximately 56 milliseconds which is 36-fold improvement comparing to 2010 milliseconds reported in prior work [41].

Figure 6.3.b is showing the average latency to obtain pseudonyms observed by vehicles. As illustrated, all vehicles received their batch of 100 pseudonyms within 4900 milliseconds. It should also be noted that this experiment is simulating a flash crowd scenario in which not only 100 new vehicles are joining the system every second, but also they keep requesting new batches of pseudonyms every 1 to 5 seconds. This means that after having all 50000 vehicles joined the system, VPKIaaS have been issuing 50000 to 250000 pseudonyms every 5 seconds. In other words, after an hour at least ≈36 million vehicles are served.

6.5 Dynamic Scalability & High Availability

In this experiment, we used Config-2 in table 6.1 with 500 pseudonyms per request to show the dynamic scalability and high availability of the VPKIaaS. Figure 6.4.a derived from monitoring system resources on the Kubernetes cluster, shows average CPU utilization by system entities (LTCA & PCA) matched with the number of requests arrived. As illustrated, increasing the number of requests, has resulted in seeing spikes at CPU utilization, but there is a fall right after each spike even though the load has been increasing with the same hatch rate. That is due to the scaling-out action. When HPA sees Pods passing the 60% threshold mentioned in section 6.4, it reacts by sending



(a) (b)

Figure 6.4: Each vehicle requests 500 pseudonyms (CPU utilization observed by HPA). (a) Number of active vehicles and CPU utilization. (b) Dynamic scalability of VPKIaaS system

[Taken from [17]]

a message to the Kubernetes scheduler asking to increase the number of Pods to be able to cope with the load. When new Pods start running, they join others by accepting traffic from the load-balancer, hence the load is shared and the spike falls below the threshold.

Figure 6.4.b shows the CPU utilization of system entities (LTCA & PCA) monitored by HPA along with the number of running Pods serving requests at each time. We see both LTCA and PCA start by running one Pod,which is their defined desired number. As HPA confirms existing Pods cannot deal with the load, more Pods join the system. Obviously, LTCA is handling less load comparing to PCA, so it can be seen that PCA is running by 80 Pods at $t = 1500$ and LTCA is running by 4 Pods. Decreasing the number of Pods at $t = 1800$ when we are stopping the experiment by switching off the load generator, shows VPKIaaS can scale in dynamically to have just enough Pods to make sure the system is functional.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this master thesis, we tackle the challenges of high-availability and dynamic scalability of the VPKI for a large-scale deployment. A state-of-the-art VPKI was refactored and architected to fit in microservices running on GCP. With this, its availability, scalability towards a cost-effective VPKI deployment was guaranteed. In addition to that, by adding health and load metrics to microservices, and leveraging an orchestration platform, we managed to add fault tolerance and self-healing features to VPKIaaS services. A secret management solution for VPKIaaS was proposed to ensure confidentiality of secrets during the deployment procedure of microservices. Also, by designing a hybrid approach of storing data, we showed that horizontal dynamic scalability is achievable while fully eradicating Sybil attacks without compromising performance. This effort would result in facilitating and catalyzing the deployment of VPKIaaS for a large-scale VC system.

7.2 Future Work

As we were conducting our experiments, we noticed a discrepancy between the number of served pseudonym requests from Locust's perspective and the number of issued pseudonyms from VPKI's perspective. We figured the reason was timeouts that are defined by default within the locust framework, meaning that requests were being served and the response was coming back to Locust but was not being counted as a successful transaction, because Locust had already considered it exceeding its timeout. By implementing more methods of the Locust interfaces, we believe we can get a more accurate results regarding the

total number of requests from the client side.

Also, in our experiments, unlike VPKIaaS that we allocated dynamic resources for, we designed the load generator to have static resources. As a result of this configuration, the load generation could not scale dynamically. In order to create the flash crowd scenario in 6.2, the load generator is also under considerable load due to the key-pair generation operations. This can be improved by making Locust dynamically scalable so we make sure more stress tests can be conducted hitting higher performance in benchmarks.

Another improvement is to implement a rate limiting technique based on puzzles, e.g., [97, 98], which can be toggled on and off automatically upon receiving higher loads suspicious to be originated as part of a DDoS attack. When it is toggled on, vehicles need to solve a puzzle in order to be able to send a request towards VPKIaaS. Such a security mechanism, can degrade the power of an attacker to a great extent, hence providing a better protection against resource exhaustion attacks.

Further improvement of the secret management mechanism, could also be a potential story to follow. Besides implementation of the hybrid solution mentioned in 4.2.4, one can explore the possibility of using Cloud HSM to offer higher level of compliance according to security standards. Although using managed KMS solutions, comes with a risk of vendor locks-in, we inspect that could be avoided by using the SOPS project [99] to have a cloud-agnostic abstraction around the KMS service.

One interesting area of work is to explore the options of using a service mesh when deploying VPKIaaS on Kubernetes. Leveraging features that a service mesh offers, we can introduce authorization policies to further protect microservices communication channels, hence controlling their ingress and egress traffic. Also we can introduce mutual TLS (mTLS) to make sure only intended messages can reach to our services. Then, researching options for federation of the service mesh is worth to be further investigated, which can potentially facilitate having a multi-cluster and/or multi-cloud solution providing even more robustness to services. In a geographically distributed multi-cluster setup, by exploiting the weighted routing functionality of Domain Name System (DNS) service, and forwarding requests to the closest Kubernetes cluster, we can optimize the round-trip of packets sent between vehicles and VPKIaaS.

Bibliography

[1] *6 firms that are testing driverless cars*. Accessed March 1, 2021. URL: https://ddswireless.com/blog/6-companies-leading-the-charge-in-self-driving-vehicles/.

[2] *Car to car communications a step closer*. 2012. URL: https://www.itsinternational.com/its10/feature/car-car-communications-step-closer.

[3] *U.S. Department of Transportation Announces Decision to Move Forward with Vehicle-to-Vehicle Communication Technology for Light Vehicles*. 2014. URL: https://mobility21.cmu.edu/u-s-department-of-transportation-announces-decision-to-move-forward-with-vehicle-to-vehicle-communication-technology-for-light-vehicles/.

[4] *Google Self-Driving Car Project*. Accessed March 1, 2021. URL: https://waymo.com/.

[5] IEEE-1609.2. "IEEE Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages". In: (2016). URL: https://standards.ieee.org/standard/1609_2-2016.html.

[6] ETSI. *ETSI TS 103 097 v1.2.1-Intelligent Transport Systems (ITS); Security; Security Header and Certificate Formats, Standard, TC ITS*. ETSI TS 103 097, 2015. URL: https://www.etsi.org/deliver/etsi_ts/103000_103099/103097/01.02.01_60/ts_103097v010201p.pdf.

[7] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions*. ETSI Tech. TR-102-638, 2009. URL: https://www.etsi.org/deliver/etsi_tr/102600_102699/102638/01.01.01_60/tr_102638v010101p.pdf.

[8] ETSI TR 102 731. *Intelligent Transport Systems (ITS); Security; Security Services and Architecture*. 2009. URL: https://www.etsi.org/deliver/etsi_ts/102700_102799/102731/01.01.01_60/ts_102731v010101p.pdf.

[9] ETSI TR 102 941. *Intelligent Transport Systems (ITS); Security; Trust and Privacy Management*. 2012. URL: https://www.etsi.org/deliver/etsi_ts/102900_102999/102941/01.01.01_60/ts_102941v010101p.pdf.

[10] *PKI Memo*. Tech. rep. C2C-CC, 2011. URL: http://www.car-2-car.org/.

[11] Panagiotis Papadimitratos, Virgil Gligor, and Jean-Pierre Hubaux. "Securing Vehicular Communications-Assumptions, Requirements, and Principles". In: *Workshop on Embedded Security in Cars (ESCAR)*. Berlin, Germany, 2006, pp. 5–14. URL: https://people.kth.se/~papadim/publications/fulltext/secure-vehicular-communication-requirements-fundamentals.pdf.

[12] Panagiotis Papadimitratos, Levente Buttyan, Tamas Holczer, Elmar Schoch, Julien Freudiger, Maxim Raya, Zhendong Ma, Frank Kargl, Antonio Kung, and J-P Hubaux. "Secure Vehicular Communication Systems: Design and Architecture". In: *IEEE Communications Magazine* 46.11 (2008), pp. 100–109. URL: https://ieeexplore.ieee.org/document/4689252.

[13] Frank Kargl, Panos Papadimitratos, Levente Buttyan, M Muter, Elmar Schoch, Bjoern Wiedersheim, Ta-Vinh Thong, Giorgio Calandriello, Albert Held, and Antonio Kung. "Secure Vehicular Communication Systems: Implementation, Performance, and Research Challenges". In: *IEEE Communications Magazine* 46.11 (2008), pp. 110–118. URL: https://ieeexplore.ieee.org/abstract/document/4689253.

[14] Panagiotis Papadimitratos, Levente Buttyan, J-P Hubaux, Frank Kargl, Antonio Kung, and Maxim Raya. "Architecture for Secure and Private Vehicular Communications". In: *IEEE International Conference on ITS Telecommunications (ITST)*. Sophia Antipolis, France, 2007, pp. 1–6. URL: https://ieeexplore.ieee.org/abstract/document/4295890.

[15] W. Whyte, A Weimerskirch, V. Kumar, and T. Hehn. “A Security Credential Management System for V2V Communications”. In: *IEEE Vehicular Networking Conference (VNC)*. Boston, MA, 2013, pp. 1–8. URL: https://ieeexplore.ieee.org/abstract/document/6737583.

[16] M. Khodaei, H. Jin, and P. Papadimitratos. “SECMACE: Scalable and Robust Identity and Credential Management Infrastructure in Vehicular Communication Systems”. In: *IEEE Transactions on Intelligent Transportation Systems* 19.5 (2018), pp. 1430–1444. URL: https://ieeexplore.ieee.org/abstract/document/8332521.

[17] Mohammad Khodaei, Hamid Noroozi, and Panos Papadimitratos. “Scaling Pseudonymous Authentication for Large Mobile Systems”. In: *Proceedings of the 12th ACM Conference on Security & Privacy in Wireless and Mobile Networks (ACM WiSec)*. Miami, FL, USA, 2019. URL: https://dl.acm.org/doi/10.1145/3317549.3323410.

[18] P. Papadimitratos, A La Fortelle, K. Evenssen, R. Brignolo, and S. Cosenza. “Vehicular Communication Systems: Enabling Technologies, Applications, and Future Outlook on Intelligent Transportation”. In: *IEEE Communications Magazine* 47.11 (2009), pp. 84–95. URL: https://ieeexplore.ieee.org/document/5307471.

[19] Reza Shokri, George Theodorakopoulos, Panos Papadimitratos, Ehsan Kazemi, and J-P Hubaux. “Hiding in the Mobile Crowd: Location Privacy through Collaboration”. In: *IEEE Transactions on Dependable and Secure Computing* 11.3 (2014), pp. 266–279. URL: https://ieeexplore.ieee.org/abstract/document/6682907.

[20] Hongyu Jin and Panos Papadimitratos. “Resilient Privacy Protection for Location-Based Services Through Decentralization”. In: *ACM Transactions on Privacy and Security (ACM TOPS)* 22.4 (2019), 21:1–36. URL: https://dl.acm.org/doi/abs/10.1145/3319401.

[21] Stylianos Gisdakis, Vasileios Manolopoulos, Sha Tao, Ana Rusu, and Panagiotis Papadimitratos. “Secure and Privacy-Preserving Smartphone-Based Traffic Information Systems”. In: *IEEE Transactions on Intelligent Transportation Systems (IEEE ITS)* 16.3 (2015), pp. 1428–1438.

[22] V. Manolopoulos, S. Tao, A. Rusu, and P. Papadimitratos. “HotMobile 2012 Demo: Smartphone-based Traffic Information System for Sustainable Cities”. In: *ACM Mobile Computing and Communications Review (ACM MC2R)* 16.4 (2012), pp. 30–31. ISSN: 1559-1662.

[23] V. Manolopoulos, Panos Papadimitratos, S. Tao, and A. Rusu. "Securing Smartphone based ITS". In: *IEEE International Conference on ITS Telecommunications (IEEE ITST)*. St. Petersburg, Russia, 2011, pp. 201–206.

[24] H. Jin, M. Khodaei, and P. Papadimitratos. "Security and Privacy in Vehicular Social Networks". In: *Vehicular Social Networks*. Taylor & Francis Group, 2016. URL: https://www.taylorfrancis.com/chapters/edit/10.1201/9781315368450-12/security-privacy-vehicular-social-networks-hongyu-jin-mohammad-khodaei-panos-papadimitratos.

[25] Virendra Kumar, Jonathan Petit, and William Whyte. "Binary Hash Tree based Certificate Access Management for Connected Vehicles". In: *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec)*. Boston, USA, 2017. URL: https://dl.acm.org/doi/abs/10.1145/3098243.3098257.

[26] Patrick McDaniel and Aviel Rubin. "A Response to *"Can We Eliminate Certificate Revocation Lists?"*" In: *FC (Springer)*. Berlin, Heidelberg, 2000, pp. 245–258. URL: https://doi.org/10.1007/3-540-45472-1_17.

[27] Jeremy Clark and Paul C Van Oorschot. "SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements". In: *IEEE SnP*. Berkeley, USA, 2013. URL: https://ieeexplore.ieee.org/abstract/document/6547130.

[28] Mohammad Khodaei and Panos Papadimitratos. "Efficient, Scalable, and Resilient Vehicle-Centric Certificate Revocation List Distribution in VANETs". In: *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (ACM WiSec)*. Stockholm, Sweden, 2018. URL: https://dl.acm.org/doi/abs/10.1145/3212480.3212481.

[29] Marcos A Simplicio Jr, Eduardo Lopes Cominetti, Harsh Kupwade Patil, Jefferson E Ricardini, and Marcos Vinicius M Silva. "ACPC: Efficient Revocation of Pseudonym Certificates using Activation Codes". In: *Elsevier Ad Hoc Networks* (2018). URL: https://www.sciencedirect.com/science/article/pii/S1570870518304761.

[30] M. Khodaei, H. Jin, and P. Papadimitratos. "Towards Deploying a Scalable & Robust Vehicular Identity and Credential Management Infrastructure". In: *IEEE Vehicular Networking Conference (VNC)*. Paderborn, Germany, 2014. URL: https://ieeexplore.ieee.org/abstract/document/7013306.

[31] Ismail Ari, Bo Hong, Ethan L Miller, Scott A Brandt, and Darrell DE Long. "Managing Flash Crowds on the Internet". In: *IEEE/ACM MASCOTS*. Orlando, FL, USA, 2003, pp. 246–249. URL: https://ieeexplore.ieee.org/abstract/document/1240667.

[32] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. *X. 509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. Tech. rep. 2013. URL: https://www.hjp.at/doc/rfc/rfc6960.html.

[33] Giorgio Calandriello, Panos Papadimitratos, Jean-Pierre Hubaux, and Antonio Lioy. "Efficient and Robust Pseudonymous Authentication in VANET". In: *ACM VANET*. NY, USA, 2007, pp. 19–28. URL: https://dl.acm.org/doi/abs/10.1145/1287748.1287752.

[34] Panos Papadimitratos, Giorgio Calandriello, Jean-Pierre Hubaux, and Antonio Lioy. "Impact of Vehicular Communications Security on Transportation Safety". In: *IEEE INFOCOM Mobile Networking for Vehicular Environments (MOVE) Workshop (IEEE MOVE)*. Phoenix, AZ, USA, 2008, pp. 1–6. URL: https://ieeexplore.ieee.org/abstract/document/4544663.

[35] Giorgio Calandriello, Panos Papadimitratos, J-P Hubaux, and Antonio Lioy. "On the Performance of Secure Vehicular Communication Systems". In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* 8.6 (2011), pp. 898–912. URL: https://ieeexplore.ieee.org/abstract/document/5611547.

[36] M. Khodaei, A. Messing, and P. Papadimitratos. "RHyTHM: A Randomized Hybrid Scheme To Hide in the Mobile Crowd". In: *IEEE Vehicular Networking Conference (VNC)*. Torino, Italy, 2017. URL: https://ieeexplore.ieee.org/abstract/document/8275642.

[37] Zhendong Ma, Frank Kargl, and Michael Weber. "Pseudonym-on-demand: A New Pseudonym Refill Strategy for Vehicular Communications". In: *IEEE VTC*. Calgary, BC, 2008, pp. 1–5. URL: https://doi.org/10.1109/VETECF.2008.455.

[38] Mohammad Khodaei and Panos Papadimitratos. “Evaluating On-demand Pseudonym Acquisition Policies in Vehicular Communication Systems”. In: *Proceedings of the First International Workshop on Internet of Vehicles and Vehicles of Internet (IoV/VoI)*. Paderborn, Germany, 2016. URL: https://dl.acm.org/doi/abs/10.1145/2938681.2938684.

[39] John R Douceur. “The Sybil Attack”. In: *ACM Peer-to-peer Systems*. London, UK, 2002. URL: https://link.springer.com/chapter/10.1007/3-540-45748-8_24.

[40] Hamid Noroozi, Mohammad Khodaei, and Panos Papadimitratos. “DEMO: VPKIaaS: A Highly-Available and Dynamically-Scalable Vehicular Public-Key Infrastructure”. In: *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (ACM WiSec)*. Stockholm, Sweden, 2018. URL: https://dl.acm.org/doi/abs/10.1145/3212480.3226100.

[41] Pierpaolo Cincilla, Omar Hicham, and Benoit Charles. “Vehicular PKI Scalability-Consistency Trade-Offs in Large Scale Distributed Scenarios”. In: *IEEE Vehicular Networking Conference (VNC)*. Columbus, Ohio, USA, 2016. URL: https://ieeexplore.ieee.org/abstract/document/7835970.

[42] Martin Fowler and Kent Beck. *Refactoring: Improving the design of existing code*. 2018. URL: https://martinfowler.com/books/refactoring.html.

[43] Mohammad Khodaei, Hamid Noroozi, and Panos Papadimitratos. “POSTER: Privacy Preservation through Uniformity”. In: *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (ACM WiSec)*. Stockholm, Sweden, 2018, pp. 279–280. URL: https://dl.acm.org/doi/abs/10.1145/3212480.3226101.

[44] *Redis, In-memory Data Structure Store, Used as a Database*. 2018. URL: https://redis.io/.

[45] *An Open Source Load Testing Tool*. 2019. URL: https://locust.io/.

[46] *What’s a Linux container?* 2021. URL: https://www.redhat.com/en/topics/containers/whats-a-linux-container.

[47] *What is a Container?A standardized unit of software*. 2021. URL: https://www.docker.com/resources/what-container.

[48] *Kubernetes: Production-Grade Container Orchestration*. 2019. URL: https://kubernetes.io/.

[49] *Google Kubernetes Engine Overview*. 2021. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview.

[50] *Kubernetes workloads, Pods*. 2021. URL: https://kubernetes.io/docs/concepts/workloads/pods/.

[51] *Kubernetes workloads, Deployments*. 2021. URL: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[52] *Kubernetes service*. 2021. URL: https://kubernetes.io/docs/concepts/services-networking/service/.

[53] *Kubernetes ingress*. 2021. URL: https://kubernetes.io/docs/concepts/services-networking/ingress/.

[54] *Kubelet*. 2021. URL: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/.

[55] Martin Fowler. *Microservices, a definition of this new architectural term*. 2014. URL: https://martinfowler.com/articles/microservices.html.

[56] T. Leinmüller, L. Buttyan, J-P. Hubaux, F. Kargl, R. Kroh, P. Papadimitratos, M. Raya, and E. Schoch. "SEVECOM-Secure Vehicle Communication". In: *Proceedings of IST Mobile Summit*. 2006. URL: https://nss.proj.kth.se/publications/fulltext/sevecom-early-1.pdf.

[57] Antonio Kung. *Security Architecture and Mechanisms for V2V/V2I, SeVeCom - Deliverable 2.1*. Version 3.0. 2008. URL: https://sevecom.eu/Deliverables/Sevecom_Deliverable_D2.1_v3.0.pdf.

[58] PRESERVE-Project. 2015. URL: www.preserve-project.eu/.

[59] Stylianos Gisdakis, Marcello Laganà, Thanassis Giannetsos, and Panos Papadimitratos. "SEROSA: SERvice Oriented Security Architecture for Vehicular Communications". In: *IEEE Vehicular Networking Conference (VNC)*. Boston, MA, USA, 2013. URL: https://ieeexplore.ieee.org/abstract/document/6737597.

[60] Stylianos Gisdakis, Thanassis Giannetsos, and Panos Papadimitratos. "SPPEAR: Security and Privacy-preserving Architecture for Participatory-sensing Applications". In: *ACM Conference on Security & Privacy in Wireless and Mobile Networks (ACM WiSec)*. Oxford, United Kingdom, 2014, pp. 39–50. ISBN: 978-1-4503-2972-9.

[61] Stylianos Gisdakis, Thanassis Giannetsos, and Panagiotis Papadimitratos. "Security, Privacy, and Incentive Provision for Mobile Crowd Sensing Systems". In: *IEEE Internet of Things Journal (IEEE IoT)* 3.5 (2016), pp. 839–853.

[62] Thanassis Giannetsos, Stylianos Gisdakis, and Panos Papadimitratos. "Trustworthy People-Centric Sensing: Privacy, Security and User Incentives Road-Map". In: *IEEE IFIP Mediterranean Ad Hoc Networking Workshop (IEEE IFIP MedHocNet)*. Piran, Slovenia, 2014, pp. 39–46.

[63] M. Khodaei and P. Papadimitratos. "Scalable & Resilient Vehicle-Centric Certificate Revocation List Distribution in Vehicular Communication Systems". In: *IEEE Transactions on Mobile Computing (TMC)* (2021). URL: https://ieeexplore.ieee.org/abstract/document/9042314.

[64] Jason J Haas, Yih-Chun Hu, and Kenneth P Laberteaux. "Efficient Certificate Revocation List Organization and Distribution". In: *IEEE Journal on Selected Areas in Communications (JSAC)* 29.3 (2011), pp. 595–604. URL: https://ieeexplore.ieee.org/abstract/document/5719271.

[65] Kenneth P Laberteaux, Jason J Haas, and Yih-Chun Hu. "Security Certificate Revocation List Distribution for VANET". In: *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking*. New York, NY, USA, 2008. URL: https://dl.acm.org/doi/abs/10.1145/1410043.1410063.

[66] Jason J Haas, Yih-Chun Hu, and Kenneth P Laberteaux. "Design and Analysis of a Lightweight Certificate Revocation Mechanism for VANET". In: *Proceedings of the sixth ACM international workshop on VehiculAr InterNETworking*. New York, NY, USA, 2009. URL: https://dl.acm.org/doi/abs/10.1145/1614269.1614285.

[67] Julien Freudiger, Maxim Raya, Márk Félegyházi, Panos Papadimitratos, and Jean-Pierre Hubaux. "Mix-zones for Location Privacy in Vehicular Networks". In: *Win-ITS*. Vancouver, BC, Canada, 2007. URL: https:

//people.kth.se/~papadim/publications/fulltext/location-privacy-mix-zones-vanet.pdf.

[68] M. Khodaei and P. Papadimitratos. "Cooperative Location Privacy in Vehicular Networks: Why Simple Mix-zones are not Enough". In: *IEEE Internet Of Things Journal* (2021). URL: https://ieeexplore.ieee.org/document/9288855.

[69] Christian Vaas, Mohammad Khodaei, Panos Papadimitratos, and Ivan Martinovic. "Nowhere to hide? Mix-Zones for Private Pseudonym Change using Chaff Vehicles". In: *IEEE Vehicular Networking Conference (VNC)*. Taipei, Taiwan, 2018. URL: https://ieeexplore.ieee.org/abstract/document/8628449.

[70] Xiaodong Lin, Xiaoting Sun, Pin-Han Ho, and Xuemin Shen. "GSIS: A Secure and Privacy-preserving Protocol for Vehicular Communications". In: *IEEE Transactions on Vehicular Technology* (2007). URL: https://ieeexplore.ieee.org/abstract/document/4357367.

[71] Rongxing Lu, Xiaodong Lin, Haojin Zhu, Pin-Han Ho, and Xuemin Shen. "ECPP: Efficient Conditional Privacy Preservation Protocol for Secure Vehicular Communications". In: *IEEE Conference on Computer Communications (INFOCOM)*. Phoenix, AZ, USA, 2008. URL: https://ieeexplore.ieee.org/abstract/document/4509774.

[72] M. Khodaei and P. Papadimitratos. "The Key to Intelligent Transportation: Identity and Credential Management in Vehicular Communication Systems". In: *IEEE Vehicular Technology Magazine* 10.4 (2015), pp. 63–69. URL: https://ieeexplore.ieee.org/abstract/document/7317862.

[73] Dave Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile*. Tech. rep. RFC 5280, 2008. URL: https://tools.ietf.org/html/rfc5280.

[74] Maximiliano Contieri. *Singleton Pattern: The Root of All Evil*. 2020. URL: https://hackernoon.com/singleton-pattern-the-root-of-all-evil-e4r3up7.

[75] *Horizontal Pod Autoscaler*. 2019. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.

[76] *YAML API Reference*. 2018. URL: https://learn.getgrav.org/advanced/yaml.

[77] *Google Cloud HSM*. 2019. URL: https://cloud.google.com/hsm/.

[78] *FIPS 140-2 Level 3 Non-Proprietary Security Policy*. 2018. URL: https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp2850.pdf.

[79] *Redis Transactions*. 2021. URL: https://redis.io/topics/transactions.

[80] *Homomorphic Encryption Standard*. 2021. URL: https://homomorphicencryption.org/.

[81] *Bluemail app removed from Google play store without notice!* 2021. URL: https://www.androidheadlines.com/2020/07/bluemail-app-removed-from-play-store-without-notice-from-google.html.

[82] L. A. Martucci, A. Zuccato, B. Smeets, S. M. Habib, T. Johansson, and N. Shahmehri. "Privacy, Security and Trust in Cloud Computing: The Perspective of the Telecommunication Industry". In: *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*. 2012, pp. 627–632. DOI: 10.1109/UIC-ATC.2012.166.

[83] *OpenStack by Open Infrastructure Foundation*. 2021. URL: https://www.openstack.org/.

[84] *Cloud Memorystore*. 2019. URL: https://cloud.google.com/memorystore/.

[85] Hsu-Chun Hsiao, Ahren Studer, Chen Chen, Adrian Perrig, Fan Bai, Bhargav Bellur, and Aravind Iyer. "Flooding-Resilient Broadcast Authentication for VANETs". In: *ACM Mobile Computing and Networking*. Las Vegas, Nevada, USA, 2011. URL: https://dl.acm.org/doi/abs/10.1145/2030613.2030635.

[86] *Cloud Armor*. 2021. URL: https://cloud.google.com/armor.

[87] Paul Heinlein. "FastCGI". In: *Linux journal* 1998.55es (1998), p. 1.

[88] *XML-RPC for C/C++*. Accessed March 1, 2021. 2021. URL: http://xmlrpc-c.sourceforge.net/.

[89] *Google Protocol Buffer*. Accessed March 1, 2021. 2021. URL: https://developers.google.com/protocol-buffers/.

[90] *Cloud SQL*. 2019. URL: https://cloud.google.com/sql.

[91] *Prometheus*. 2019. URL: https://prometheus.io/.

[92] *Grafana*. 2019. URL: https://grafana.com/.

[93] *Styx*. 2020. URL: https://github.com/go-pluto/styx.

[94] Sandesh Uppoor, Oscar Trullols-Cruces, Marco Fiore, and Jose M Barcelo-Ordinas. "Generation and Analysis of a Large-scale Urban Vehicular Mobility Dataset". In: *IEEE Transactions on Mobile Computing* 13.5 (2014), pp. 1061–1075. URL: https://ieeexplore.ieee.org/abstract/document/6468040.

[95] Lara Codeca, Raphaël Frank, and Thomas Engel. "Luxembourg SUMO Traffic (LuST) Scenario: 24 Hours of Mobility for Vehicular Networking Research". In: *IEEE Vehicular Networking Conference (VNC)*. Kyoto, Japan, 2015, pp. 1–8. URL: https://ieeexplore.ieee.org/abstract/document/7385539.

[96] John Harding, Gregory Powell, Rebecca Yoon, Joshua Fikentscher, Charlene Doyle, Dana Sade, Mike Lukuc, Jim Simons, and Jing Wang. *V2V Communications: Readiness of V2V Technology for Application*. Tech. rep. U.S. Department of Transportation - National Highway Traffic Safety Administration - DOT HS 812 014, 2014. URL: http://www.nhtsa.gov/staticfiles/rulemaking/pdf/V2V/Readiness-of-V2V-Technology-for-Application-812014.pdf.

[97] M. Abliz and T. Znati. "A Guided Tour Puzzle for Denial of Service Prevention". In: *IEEE ACSAC*. Honolulu, HI, 2009, pp. 279–288. URL: https://doi.org/10.1109/ACSAC.2009.33.

[98] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. "DoS-Resistant Authentication with Client Puzzles". In: *Proceedings of Security Protocols Workshop*. New York, USA, 2001. URL: https://doi.org/10.1007/3-540-44810-1_22.

[99] *SOPS: Secrets OPerationS*. 2021. URL: https://github.com/mozilla/sops.